Algorithms for Argument Systems

Thesis submitted in accordance with the requirements of
the University of Liverpool for the degree of Doctor in Philosophy
by
Samer Nofal

April 2013

Abstract

Argument systems are computational models that enable an artificial intelligent agent to reason via argumentation. Basically, the computations in argument systems can be viewed as search problems. In general, for a wide range of such problems existing algorithms lack five important features. Firstly, there is no comprehensive study that shows which algorithm among existing others is the most efficient in solving a particular problem. Secondly, there is no work that establishes the use of cost-effective heuristics leading to more efficient algorithms. Thirdly, mechanisms for pruning the search space are understudied, and hence, further pruning techniques might be neglected. Fourthly, diverse decision problems, for extended models of argument systems, are left without dedicated algorithms fine-tuned to the specific requirements of the respective extended model. Fifthly, some existing algorithms are presented in a high level that leaves some aspects of the computations unspecified, and therefore, implementations are rendered open to different interpretations. The work presented in this thesis tries to address all these concerns.

Concisely, the presented work is centered around a widely studied view of what computationally defines an argument system. According to this view, an argument system is a pair: a set of abstract arguments and a binary relation that captures the conflicting arguments. Then, to resolve an instance of argument systems the acceptable arguments must be decided according to a set of criteria that collectively define the argumentation semantics. For different motivations there are various argumentation semantics. Equally, several proposals in the literature present extended models that stretch the basic two components of an argument system usually by incorporating more elements and/or broadening the nature of the existing components. This work designs algorithms that solve decision problems in the basic form of argument systems as well as in some other extended models. Likewise, new algorithms are developed that deal with different argumentation semantics. We evaluate our algorithms against existing algorithms experimentally where sufficient indications highlight that the new algorithms are superior with respect to their running time.

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgement

This work has been completed with the assistance of several parties. I would like to thank my supervisors, Paul Dunne and Katie Atkinson, who offered me great help and endless support throughout this study. I am thankful to the staff at the computer science department of University of Liverpool who put every effort in helping me to overcome many obstacles in the course of my research. Especially I give thanks to my advisors: Trevor Bench-Capon, Grant Malcolm and Russel Martin for their comments and feedback that improved my research. I am grateful to Anthony Hunter and Trevor Bench-Capon for reading my thesis and for their helpful comments that improved the presentation of this work. Finally, I appreciate my sponsor, the German-Jordanian University, for giving me the opportunity to do my PhD research without any financial hardship.

Chapter 1

Introduction

This chapter outlines the contributions and presents preliminaries. In section 1.1 argument systems are introduced. In section 1.2 the contributions of this research are highlighted. Section 1.3 discusses the role of experimental analysis in evaluating the performance of algorithms in argument systems. Section 1.4 recalls the formal definition of argument systems and gives an overview of a selection of prevalent argumentation semantics for which this work develops new algorithms. Section 1.5 presents an introduction to *value based* argument systems. In section 1.6 argument systems with *recursive attacks* are described briefly. Section 1.7 gives an overview of the existing algorithms that are closely related to the algorithms designed in this work. Lastly, section 1.8 illustrates the structure of the dissertation.

1.1 Applications of Computational Argumentation

Humans, by nature, are able to reason through arguing about their opinions, decisions and actions. An AI system might be able to reason by incorporating computational mechanisms for argumentation: so-called argument systems. An argument system is a reasoning model that is likely to be a mainstay in the study of other areas such as:

- Decision support systems (see e.g. [5, 62]).
- Agent interaction in multi agent systems (see e.g. [73, 20]).
- Legal reasoning (see e.g. [14, 89]).
- Merging conflicting knowledge bases (see e.g. [23, 3]).
- Machine learning (see e.g. [79]).

A comprehensive review of the topic of computational argumentation is given in [15], [18] and [90]. This research is centered around a widely studied model of computational argumentation put forward by Dung [37] where he abstractly views an argument system as a pair: set of arguments and a binary relation to represent the conflicting

arguments. The importance of this model is attributed to its abstract nature that facilitates the focus on the calculus of the "acceptability" of arguments, thereby providing a simple, unified way for automating argument-based reasoners. Section 1.4 recalls the definition of argument systems of [37] along with the related argumentation semantics.

1.2 Research Contributions

The contributions of this research are:

1. Engineering algorithms for decision problems in Dung's abstract argument system [37] under various argumentation semantics taking into account the subsequent points.

2. Developing concrete algorithms that are more efficient than the existing algorithms of [28, 35, 78, 95, 97], and verifying performance improvements by conducting empirical evaluation and presenting analytical comparisons with existing algorithms.

3. Computations in argument systems can be seen primarily as search problems. The search space can be visualized as a tree where every node in the tree represents a different state of an argument system. The root of the tree is the initial state while zero or more of the leaf nodes represent goal states, which correspond to solution(s) to some decision problem. Figure 1.1 shows a binary search tree. The first efficiency matter considered is the nature of the search tree. That is, the branching factor and the tree depth. Broadly speaking, the smaller the branching factor and/or the shorter tree depth, the better performance is attained. These two aspects form an important measure to distinguish between various algorithms. Thus, new algorithms are designed that work in an efficient tree structure. One determinant factor is to define a powerful mechanism through which an argument system transitions from a state to another. This is exactly what the new algorithms achieve. Lastly, we state that depth-first search (DFS) is the adopted traverse method in this work for two reasons:

   (a) DFS allows the use of heuristics while breadth-first search (BFS) does not.

   (b) Broadly speaking, space complexity of DFS procedures grows polynomially while BFS requires space in exponential order.

4. The goal state(s) might be anywhere in the search tree. So, one might wonder whether or not there is a cost-effective way to predict the likelihood of a goal state to be encountered if a branch is proceeded, and thus, to allow the search process to prioritize following a branch over another, based on that speculation. This is the second efficiency matter this research examines; we describe powerful

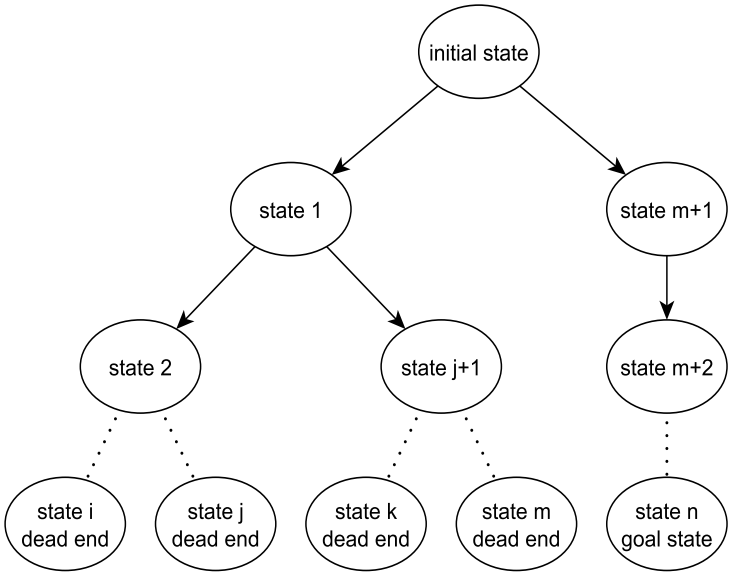

Figure 1.1: Visualizing the search space as a tree such that every node represents a state of the underlying argument system.

heuristics that might guide the search process to the shortest path to the goal state. This is particularly important when the concerned decision problem is around finding one solution rather than all solutions.

5. As not all leaf nodes are goal states, it will be an unproductive use of time to explore some subtrees. Consequently, an important feature in designing a search algorithm is to find out which properties of a search state would detect such fruitless subtrees, and thereby, the search process avoids expanding them. This is the third performance issue we addressed; we specify effective pruning strategies for a wide range of decision problems in argument systems.

6. The algorithms of this work are introduced in full and unambiguously. It might be the case that some algorithms in the literature are presented in a high level, and therefore, implementations of some computations are left open to different interpretations. This work designs meticulous procedures while keeping formality in all aspects.

7. In the literature there are several extended models of Dung's argument system. In this research it is argued that algorithms for decision problems in such extended models are understudied. Subsequently, algorithms are built for two examples of such models: value based argument systems [13] and argument systems with recursive attacks [7]. The developed algorithms take into account the issues discussed earlier in the previous points.

1.3 Research Methodology

The importance of experimental evaluation for algorithms in argument systems can be summarized as three benefits:

1. Theoretical evaluation might not reveal which algorithm among others is most efficient in solving a given problem practically, while empirical analysis might be a useful methodology in this matter. For example, the dominant simplex algorithm for linear programming runs efficiently in practice although it has an exponential worst-case time complexity.

2. Experimental analysis often provides a means to evaluate the average practical behavior of algorithms. This is already exploited in studying the behavior of algorithms solving the satisfiability problem.

3. As stated earlier, computations in argument systems can be seen as search problems, and so, it is essential to measure empirically the effectiveness of the applied heuristics.

Hence, in addition to analyzing our algorithms in comparison to existing algorithms we conduct experiments to verify the efficiency gain of the new algorithms. To this end, we track the average elapsed time of the concerned algorithms running over randomly generated argument systems, using the reported running times as a basis to compare between algorithms. A full description of experiments settings will be presented within the thesis. But next we provide some necessary background material.

1.4 Argument Systems: Preliminaries

An argument system, as defined in [37], is a pair (A, R) where A is a set of arguments and $R \subseteq A \times A$ is a binary relation called the *attack* relation. We refer to $(x, y) \in R$ as x attacks y (or y is attacked by x). We denote by $\{x\}^-$ respectively $\{x\}^+$ the subset of A containing those arguments that attack (resp. are attacked by) the argument x, extending this notation in the natural way to *sets* of arguments, so that for $S \subseteq A$,

$$\begin{aligned} S^- &= \{\, y \in A \,:\, \exists\, x \in S \text{ s.t. } y \in \{x\}^-\} \\ S^+ &= \{\, y \in A \,:\, \exists\, x \in S \text{ s.t. } y \in \{x\}^+\} \end{aligned}$$

An argument x is *acceptable* w.r.t. $S \subseteq A$ if and only if for every $(y, x) \in R$, there is some $z \in S$ for which $(z, y) \in R$. A subset $S \subseteq A$ is *conflict free* if and only if for each $(x, y) \in S \times S$, $(x, y) \notin R$. A subset $S \subseteq A$ is *admissible* if and only if it is conflict free and every $x \in S$ is acceptable w.r.t. S. A *preferred* extension is a maximal (w.r.t. $\subseteq$) admissible set. Focusing on preferred semantics, an argument is *skeptically* accepted if and only if the argument is in *all* preferred extensions. On the other hand, an argument is *credulously*

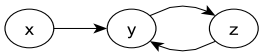

Figure 1.2: An argument system.

accepted if and only if the argument is in at least one preferred extension. A set $S \subseteq A$ is *stable* iff S is conflict free and $S^+ = A \setminus S$. A subset $S \subseteq A$ is a *complete* extension if and only if S is admissible such that for each x acceptable w.r.t. S, $x \in S$. For $S \subseteq A$, the *range* of S is defined as $S \cup S^+$. Then, S is a *stage* extension if and only if S is a conflict free set with a maximal (w.r.t. $\subseteq$) range. A set $S \subseteq A$ is a *semi stable* extension if and only if S is an admissible set with a maximal (w.r.t. $\subseteq$) range. A set $S \subseteq A$ is *ideal* if and only if S is admissible and for every preferred extension $pref$, $S \subseteq pref$. Ideal semantics [39] provides a *unique* extension called the ideal extension, which is the maximal (w.r.t. $\subseteq$) ideal set. Coming to *grounded* semantics, let $F : 2^A \rightarrow 2^A$ be a function that, given $S \subseteq A$, returns the set of acceptable arguments w.r.t. S. Then the grounded extension is the least fixed point of F. Preferred, complete, stable and grounded semantics are introduced in [37], whereas ideal semantics, stage semantics and semi stable semantics are defined in [39, 96, 24] respectively. For instance, consider the argument system depicted in figure 1.2 where nodes represent arguments and edges correspond to attacks (i.e. elements in R). For this argument system $\{x, z\}$ is the preferred, grounded, stable, ideal, complete, semi stable and stage extension. Note that we do not mean by this example to show differences between semantics. For an excellent introduction to argumentation semantics see [6].

1.5 Value Based Argument Systems: Preliminaries

To consider mechanisms with which to model persuasive argument in practical reasoning (i.e. reasoning about what to do, which is distinguished from reasoning about what to believe) Bench-Capon [13] extends Dung's system to accommodate the notion of arguments promoting "social values". Bench-Capon's value based argument system is a 4-tuple (A, R, V, η) where A is a set of arguments, $R \subset A \times A$ is a binary relation, V is a non-empty set of social values, $\eta : A \rightarrow V$ maps arguments in A to values in V. A total ordering, α of V is referred to as a *specific audience*. Given a specific audience, α, we refer to $(v_i, v_j) \in \alpha$ as "v_i is preferred to v_j" or "$v_i \succ v_j$". We denote the set of all specific audiences by $\mathcal{U}$. Audiences offer a means to distinguish attacks $(x, y) \in R$ which do not succeed as a consequence of expressed value priorities recognizing that different audiences may have different interests and aspirations. Formally, we say x *defeats* y w.r.t. the *audience* α if and only if $(x, y) \in R$ and $(\eta(x), \eta(y)) \in \alpha$. An argument x is acceptable for an audience α w.r.t. $S \subseteq A$ if and only if for every y that defeats x (w.r.t. α) there is some $z \in S$ that defeats y w.r.t. α. A set $S \subseteq A$ is conflict free for

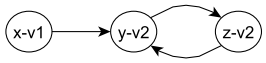

Figure 1.3: A value based argument system.

the audience α if and only if for all $x, y \in S$ it is not the case that x defeats y w.r.t. α. Similarly, S is admissible under α if and only if it is conflict free under α and every $x \in S$ is acceptable for α w.r.t. S. The α-preferred extensions are the maximal (w.r.t. $\subseteq$) admissible under α sets. An argument x is *objectively* accepted if and only if for *every* α, x is in *every* α-preferred extension[1]. On the other hand, x is *subjectively* accepted if and only if there is some α for which x is in an α-preferred extension. For instance, consider the value based argument system in figure 1.3 where $A = \{x, y, z\}$, $R = \{(x, y), (y, z), (z, y)\}$, $V = \{v_1, v_2\}$ and $\eta = \{(x, v_1), (y, v_2), (z, v_2)\}$. The nodes in figure 1.3 are labelled by argument-value identifiers. If $v_1 \succ v_2$ the $(v_1 \succ v_2)$-preferred extension is $\{x, z\}$ otherwise the $(v_2 \succ v_1)$-preferred extensions are $\{\{x, y\}, \{x, z\}\}$. Therefore, x is objectively accepted while y and z are subjectively accepted.

1.6 Argument Systems with Recursive Attacks: Preliminaries

To provide an explicit way to weaken an attack the formalisms of [76, 57, 7] extend Dung's argumentation system such that attacks (i.e. elements of R) are subject to attacks themselves. In chapter 6 we develop algorithms for an instance of such formalisms: argument systems with recursive attacks introduced in [7].

An argument system with recursive attacks is a pair $(A, \overline{R})$ where A is a set of arguments and $\overline{R}$ is a set of pairs (x, y) such that $x \in A$ and ($y \in A$ or $y \in \overline{R}$). Let $x = (y, z) \in \overline{R}$ then we say that y is the source of x, denoted as $src(x) = y$, and z is the target of x, denoted as $trg(x) = z$. Let $x \in A \cup \overline{R}$ and $y \in \overline{R}$ then we say that y *directly defeats* x if and only if $x = trg(y)$. Let $x, y \in \overline{R}$ then we say y *indirectly defeats* x if and only if $src(x) = trg(y)$. Let $x \in A \cup \overline{R}$ and $y \in \overline{R}$, we say y *defeats* x if and only if y directly or indirectly defeats x.

A subset $S \subseteq A \cup \overline{R}$ is conflict free if and only if there does not exist $(x, y) \in S \times S$ with x *defeats* y. An element $x \in A \cup \overline{R}$ is acceptable w.r.t. $S \subseteq A \cup \overline{R}$ if and only if for each $y \in \overline{R} : y$ *defeats* x, there is some $z \in S : z$ *defeats* y. A subset $S \in A \cup \overline{R}$ is admissible if and only if S is conflict free and for each $x \in S$, x is acceptable w.r.t. S. A preferred extension is a maximal (w.r.t. $\subseteq$) admissible set. A subset $S \subseteq A \cup \overline{R}$ is a stable extension if and only if S is conflict free and for each $x \in A \cup \overline{R} : x \notin S$, there exists $y \in S : y$ *defeats* x. A subset $S \subseteq A \cup \overline{R}$ is a complete extension if and only if S is admissible and every element of $A \cup \overline{R}$ which is acceptable w.r.t. S belongs to S. For $S \subseteq A \cup \overline{R}$, the range of

[1]Note that under the assumption of [13] such that directed cycles of arguments involve at least two distinct values, the α-preferred extension is *unique*.

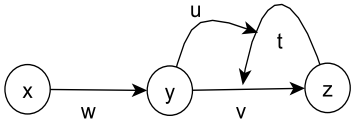

Figure 1.4: An argument system with recursive attacks.

S is defined as $S \cup S^+$, where $S^+ = \{x \in A \cup \overline{R} \mid \exists y \in S : y \; defeats \; x\}$. A subset $S \subseteq A \cup \overline{R}$ is a stage extension if and only if S is a conflict free set with a maximal (w.r.t. $\subseteq$) range. A subset $S \subseteq A \cup \overline{R}$ is a semi stable extension if and only if S is an admissible set with a maximal (w.r.t. $\subseteq$) range. An ideal set is an admissible set S such that for every preferred extension $pref$, $S \subseteq pref$. The ideal extension is the maximal (w.r.t. $\subseteq$) ideal set. Coming to grounded semantics, let $F : 2^{A \cup \overline{R}} \rightarrow 2^{A \cup \overline{R}}$ be a function that, given $S \subseteq A \cup \overline{R}$, returns {x | x is acceptable w.r.t. S}. Then the grounded extension is the least fixed point of F. Referring to figure 1.4, $\{x, z, w, t\}$ is the grounded, stable, preferred, ideal, complete, stage and semi stable extension.

1.7 Related Algorithms

Here we highlight the closely related works that present algorithms for deciding some of the problems addressed in this thesis. Specifically, we evaluate the new algorithms by comparing with:

- The algorithm of Doutre and Mengin [35] for enumerating preferred extensions.
- The algorithms of Cayrol et al. [28] for deciding credulous/skeptical acceptance under preferred semantics.
- The algorithm of Verheij [97] for deciding credulous acceptance under preferred semantics.
- The algorithm of Modgil and Caminada [78] for enumerating preferred extensions.
- The algorithms of Thang et al. [95] for deciding credulous/skeptical acceptance under preferred semantics.

Nonetheless, in every chapter we discuss, occasionally in the summary section, other previous works that are related to the content of the respective chapter. Moreover, in chapter 7 we dedicated a section for reviewing other available literature that is connected to the present work in the general sense.

1.8 Dissertation Structure

The rest of this dissertation is organized as follows:

- Chapter 2 motivates empirical analysis as a methodology for this research where we undertake experiments to evaluate the efficiency of algorithms that decide arguments' acceptance in the extended argument systems of [76] and value based argument systems. The content of this chapter is presented in [85, 84].

- In chapter 3 we present a new algorithm for enumerating preferred extensions. We show that the new algorithm is faster than the existing algorithms of [35, 78]. Likewise, we present new algorithms for deciding skeptical/credulous acceptance. Then we demonstrate that the new algorithms are more efficient than the existing algorithms of [28, 95, 97]. The content of this chapter is presented in [83, 86].

- In chapter 4 we design algorithms for stable semantics, ideal semantics, semi stable semantics , stage semantics and complete semantics. Also, we present an implementation of the algorithm of [78] for grounded semantics.

- In chapter 5 we engineer a novel algorithm for enumerating preferred extensions of a value based argument system. Afterwards, we build algorithms for deciding objective/subjective acceptance. The content of this chapter is presented in [83].

- In chapter 6 we show how to enumerate different kinds of extensions for an instance of formalisms that allow attacks on attacks: namely argument systems with recursive attacks [7].

- Chapter 7 concludes the thesis by reviewing the contributions, discussing further related works and setting up an agenda for future research.

Chapter 2

Experimental Analysis of Algorithms in Argument Systems

From theoretical computational perspectives, decision problems in argument systems are either polynomial solvable or intractable. To investigate practical efficiency, theoretical evaluation of applied algorithms does not necessarily reveal performance dissimilarities. More specifically, theoretical analysis of algorithms often draw the behavior in worst-case scenarios, which can be uncommon, while the average behavior is left uncaught. Further, the dominant asymptotic-based analysis pictures the growth trend of the running time, and in turn, does not trace the exact performance. Thus, theoretical evaluation might not reveal which algorithm among others is most efficient in solving a given problem practically [74]. Therefore, the main purpose of this chapter is to motivate the role of experimental evaluation in analyzing algorithms' behavior for decision problems in argument systems where theoretical analysis might be of little help. To this end, we pick two extended argument systems, which are basically derived from Dung's argumentation system, as a case study to empirically examine the efficiency of algorithms related to the acceptance of arguments. In particular, in section 2.1 we introduce a case in which we study experimentally three different algorithmic methods that decide arguments' acceptability in Modgil's argument system[1] [76]. Afterwards, in section 2.2 we give another case where we analyse empirically algorithms for objective/subjective acceptance in value based argument systems[2]. Finally in section 2.3 we close the chapter offering further discussions and related works.

[1] In his paper [76] Modgil refers to his model as "extended argumentation frameworks".

[2] In this chapter we discuss algorithms in value based argument systems under the assumption of multi-value cycles [13], i.e. there does not exist a cycle such that all arguments in the cycle promote the same social value, while in chapter 5 we present labeling algorithms for enumerating preferred extensions of any value based argument system.

2.1 Experimental Algorithms for Modgil's Argument System

In the context of practical reasoning, Modgil [76] elaborated a way to reason about preferences in argument systems, and thus, introduced an extended formalism of Dung's argument system.

Definition 1. *A Modgil argument system is a 3-tuple* (A,R,D) *such that* A *is a set of arguments,* $R \subseteq A \times A$, $D \subseteq A \times R$ *and if* $(x,(y,z)),(x',(z,y)) \in D$ *then* $(x,x'),(x',x) \in R$. *We refer to* $(x,(y,z)) \in D$ *as* x *attacks* (y,z) *or* (y,z) *is attacked by* x.

Referring to the Modgil argument system in figure 2.1, $A = \{w,x,y,z\}$, $R = \{(w,x),(w,z),(x,z),(y,w),(z,y)\}$, $D = \{(w,(x,z)),(y,(w,x)),(z,(w,z)),(z,(y,w))\}$. In Modgil's argument system not all attacks are *defeats*, an attack is considered a defeat w.r.t. to a set of arguments if and only if the attack is not attacked by any member of that set.

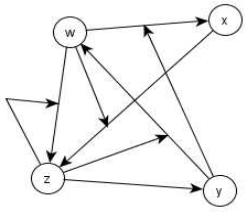

Figure 2.1: An instance of Modgil's argument system that is referenced throughout this section.

Definition 2. *Let* (A,R,D) *be a Modgil argument system and* $S \subseteq A$. *Then* $x\ defeats_S\ y$ *if and only if* $(x,y) \in R$ *and* $\nexists z \in S$ *s.t.* $(z,(x,y)) \in D$.

In Modgil's notation, $y \rightharpoonup z$ denotes $(y,z) \in R$, $x \twoheadrightarrow (y \rightharpoonup z)$ denotes $(x,(y,z)) \in D$ and $y \rightarrow^S z$ denotes $y\ defeats_S\ z$. Now we come to conflict freeness in Modgil's argument system. A set of arguments is conflict free if there is no two members that attack each other and there is no member that defeats another w.r.t. to the set itself.

Definition 3. *Let* (A,R,D) *be a Modgil argument system. Then* $S \subseteq A$ *is conflict free if and only if* $\forall x,y \in S : if (x,y) \in R$, *then* $(y,x) \notin R$ *and* $\exists z \in S$ *s.t.* $(z,(x,y)) \in D$.

Referring to the Modgil argument system in figure 2.1, $\{w,z\}$ is conflict free while $\{x,z\}$ is not. The acceptability in Modgil's argument system does not only require defending an argument but it needs also to check for the reinstatement of that defense.

Definition 4. *Let* $S \subseteq A$ *in* (A,R,D). *Then* $R_S = \{x_1 \rightarrow^S y_1,...,x_n \rightarrow^S y_n\}$ *is a reinstatement set for* $w \rightarrow^S z$, *if and only if:*

1. $w \rightarrow^S z \in R_S$,

2. *for* $i = 1...n, x_i \in S$,

3. $\forall x \rightarrow^S y \in R_S, \forall y'$ *s.t.* $(y', (x, y)) \in D$, *there is a* $x' \rightarrow^S y' \in R_S$.

Definition 5. *Let* (A, R, D) *be a Modgil argument system. Then* $y \in A$ *is acceptable w.r.t.* $S \subseteq A$, *if and only if* $\forall x$ *s.t.* $x \rightarrow^S y, \exists z \in S$ *s.t.* $z \rightarrow^S x$ *and there is a reinstatement set for* $z \rightarrow^S x$.

Referring to the Modgil argument system in figure 2.1, y is acceptable w.r.t. $\{x, y\}$ and the reinstatement set for $x \rightarrow^S z$ is $\{x \rightarrow^S z, y \rightarrow^S w\}$. A subset $S \subseteq A$ is an admissible set if and only if S is conflict free and for each $x \in S : x$ is acceptable w.r.t. S. A set $S \subseteq A$ is a preferred extension if and only if S is a set inclusion maximal admissible set. Referring to the Modgil argument system in figure 2.1, $\{x, y\}$ and $\{w, z\}$ are preferred extensions. With respect to preferred semantics, an argument might be in all extensions, and so, it is skeptically accepted. On the other hand, an argument might be in at least one preferred extension, and hence, it is credulously accepted. In the next section we offer basic methods that could be used in computing skeptical/credulous acceptance.

2.1.1 Algorithms for Acceptability

In this section we present three different algorithmic methods for deciding the acceptability of an argument w.r.t. to a set of arguments. The first method is taken from [47] while we formulate the other two methods. The three methods result in algorithms that run in polynomial time [47]. Nevertheless, experiments in section 2.1.2 suggest that the algorithms show differences in practical behavior.

2.1.1.1 Method 1

[47] define an algorithm for deciding the acceptability of arguments in Modgil's argument system, see algorithm 1. To exemplify, consider the Modgil argument system in figure 2.1. Then to decide the acceptability of y w.r.t. $\{x, y\}$ algorithm 1 removes $w \rightharpoonup x$ (line 1) and then colors $z \rightharpoonup y$ blue (line 2). Afterwards, it colors $x \rightharpoonup z$ red (line 5) and subsequently $w \twoheadrightarrow (x \rightharpoonup z)$ blue (line 7). Next, it colors $y \rightharpoonup w$ red (line 5) and later $z \twoheadrightarrow (y \rightharpoonup w)$ blue (line 7). At this stage (line 16), algorithm 1 finds y attacked by z, but z is attacked by x. Hence, y is acceptable w.r.t. $\{x, y\}$. Note that lines 9-15 do not do any action in this example.

2.1.1.2 Method 2

As we could say that an argument is acceptable w.r.t. a set of arguments, we might also say that an argument is *refutable* w.r.t. a set of arguments if the argument is defeated by a member of the respective set and there is a reinstatement set for that defeat.

Algorithm 1: Deciding the acceptability of $x \in A$ w.r.t. $S \subseteq A$ in a Modgil argument system (A, R, D).

1: $R \leftarrow R \setminus \{(y,z) : y \twoheadrightarrow^S z\}$
2: color each attack $y \rightarrow^S x$ BLUE
3: **repeat**
4: **for all** $y \in A$ s.t. $y \rightarrow^S x$ or $(y,(u,v))$ is BLUE **do**
5: color $z \rightarrow^S y$ RED for each $z \in S$ s.t. $z \rightarrow^S y$
6: **for all** $z \rightarrow^S y$ colored RED **do**
7: color each attack $(v,(z,y)) \in D$ BLUE
8: **until** no change in attack colors
9: **repeat**
10: **if** $\exists y \in A$ s.t. $(y,(v,w))$ is BLUE and there is no $u \rightarrow^S y$ colored RED **then**
11: $R \leftarrow R \setminus \{(v,w)\}$
12: $D \leftarrow D \setminus \{(y,(v,w))\}$
13: **if** $\exists z \in S$ s.t. $((z,y)$ is RED with $(y,(u,v))$ BLUE) and $\nexists (p,(z,y)) \in D$ **then**
14: $D \leftarrow D \setminus \{(y,(u,v))\}$
15: **until** no change in D
16: **Report** whether x is acceptable w.r.t. S in the (A,R)

Definition 6. *Let (A,R,D) be a Modgil argument system. Then $x \in A$ is refutable w.r.t. $S \subseteq A$, if and only if $\exists y \in S$ s.t. $y \rightarrow^S x$ and there is a reinstatement set for $y \rightarrow^S x$.*

Consider the Modgil argument system in figure 2.1, x is refutable w.r.t. $S = \{w,z\}$ since $w \rightarrow^S x$ and $\{w \rightarrow^S x, z \rightarrow^S y\}$ is the reinstatement set. To decide whether an argument is acceptable to a set of arguments or not, method 2 examines the attackers who defeat the argument w.r.t. the concerned set. If all defeaters are refutable then method 2 asserts acceptability otherwise it halts and declines acceptability. Method 2 decides that an argument is refutable w.r.t. a set of arguments if:

1. there is a member of the respective set that defeats the argument in question, and
2. all arguments which attack the defense in (1), are also refutable w.r.t. the respective set.

Method 2 is represented by algorithms 2-4. Algorithm 2 traverses the defeaters of the argument in question, and then, checks the refutability of each defeater by invoking algorithm 3. Actually, algorithm 3 calls algorithm 4 and vice versa where the objective of both algorithms is to find out the refutability of an argument. These algorithms make use of a simple coloring scheme. The default color is green while the red color is given to any argument or attack that is currently under processing to prevent infinite recursion; upon finishing the green color is returned.

As an example, consider the Modgil argument system in figure 2.1. To decide whether the conflict free $S = \{x,y\}$ is an admissible set or not, we need to check the acceptability of x and y w.r.t. to S. For x, the only attacker $w \twoheadrightarrow^S x$ and so x is acceptable.

For y, $z \rightarrow^S y$, and therefore, algorithm 2 calls algorithm 3 (line 3) to decide whether z is refutable w.r.t. S or not. At this point, algorithm 3 finds $x \rightarrow^S z$, and so, calls algorithm 4 (line 8) to see whether this defense is reinstated. Indeed, algorithm 4 finds that $w \twoheadrightarrow (x \rightharpoonup z)$ (line 2). Thereafter, algorithm 4 (line 4) invokes algorithm 3 to decide whether w is refutable or not. Actually, $y \rightarrow^S w$ (line 8) and so algorithm 3 calls algorithm 4 that finds $z \twoheadrightarrow (y \rightharpoonup w)$ (line 2). As z is RED (i.e. visited), this signals that w and z are refutable w.r.t. S and subsequently y is acceptable w.r.t. S. In what follows we present the proof of algorithms 2-4.

Proposition 1. *Let (A, R, D) be a Modgil argument system, $S \subseteq A$, $x \in A$ and $(y, x) \in R$. Then:*

1. *x is acceptable w.r.t. S if and only if algorithm 2 returns true.*
2. *x is not acceptable w.r.t. S if and only if algorithm 2 returns false.*
3. *x is refutable w.r.t. S if and only if algorithm 3 returns true.*
4. *x is not refutable w.r.t. S if and only if algorithm 3 returns false.*
5. *Let $y \rightarrow^S x$, then there is a reinstatement set for $y \rightarrow^S x$ if and only if algorithm 4 returns true.*
6. *Let $y \rightarrow^S x$, then there is no reinstatement set for $y \rightarrow^S x$ if and only if algorithm 4 returns false.*

Proof:

1. By definition 5, if x is acceptable w.r.t. S then for each $y \rightarrow^S x$, there is $z \in S$ s.t. $z \rightarrow^S y$ and there is a reinstatement set for $z \rightarrow^S y$. Algorithm 2 returns true (line 5) if for every $y \rightarrow^S x$, y is refutable w.r.t. S (line 3), which implies (by definition 6) there exists $z \in S$ with $z \rightarrow^S y$ and there is a reinstatement for $z \rightarrow^S y$.

2. x is not acceptable w.r.t. S implies that there exists $y \rightarrow^S x$ s.t. for every $z \in S$, $z \nrightarrow^S y$ or there is no reinstatement set for $z \rightarrow^S y$ (definition 5). Algorithm 2 returns false (line 4) if there is $y \rightarrow^S x$ (line 2) while y is not refutable w.r.t. S (line 3), being not refutable means (by definition 6) for every $z \in S$, $z \nrightarrow^S y$ or there is no reinstatement set for $z \rightarrow^S y$.

3. x is refutable w.r.t. S implies there exists $y \in S$ s.t. $y \rightarrow^S x$ and there is a reinstatement set for $y \rightarrow^S x$ (definition 6). Algorithm 3 returns true (lines 6 and 10) if there is $y \in S : y \rightarrow^S x$ and there is a reinstatement set for $y \rightarrow^S x$ (line 8).

4. If x is not refutable w.r.t. S then for each $y \in S$, $y \nrightarrow^S x$ or there is no reinstatement set for $y \rightarrow^S x$ (definition 6). Algorithm 3 returns false (line 12) if and only if for all $y \in S$, $y \nrightarrow^S x$ or there is no reinstatement set for $y \rightarrow^S x$.

5. Algorithm 4 returns true (line 8) if for every $z \twoheadrightarrow (y \rightharpoonup x)$ (line 2), z is refutable w.r.t. S (line 4), which means (by definition 6) there is $w \in S$ with $w \rightarrow^S z$ and there is a reinstatement set for $w \rightarrow^S z$.

6. Algorithm 4 returns false (line 6) if there exists $z \twoheadrightarrow (y \rightharpoonup x)$ (line 2) and z is not refutable w.r.t. S (line 4), which means for each $w \in S$, $w \nrightarrow^S z$ or there is no reinstatement set for $w \rightarrow^S z$ (definition 6).

■

Algorithm 2: Acceptable($x \in A$, $S \subseteq A$)

```
1:  for all (y,x) ∈ R do
2:      if y →^S x then
3:          if Refutable(y,S) = false then
4:              return false
5:  return true
```

Algorithm 3: Refutable($x \in A$, $S \subseteq A$)

```
1:  color x RED
2:  for all (y,x) ∈ R do
3:      if y ∈ S then
4:          if (y,x) is RED then
5:              color x GREEN
6:              return true
7:          else
8:              if y →^S x ∧ Reinstated((y,x),S) = true then
9:                  color x GREEN
10:                 return true
11: color x GREEN
12: return false
```

2.1.1.3 Method 3

In method 2, there are 3 algorithms for 3 tasks. From an algorithmic design point of view, that architecture can be reasonable, because, checking refutability, which is the job of algorithm 3, is a common process between algorithms 2 and 4. Nonetheless, method 3 is composed of two algorithms that fulfill the task of the three algorithms of

Algorithm 4: Reinstated($(y,x) \in R$, $S \subseteq A$)

```
1: color (y,x) RED
2: for all (z,(y,x)) ∈ D do
3:    if z is GREEN then
4:       if Refutable(z,S) = false then
5:          color (y,x) GREEN
6:          return false
7: color (y,x) GREEN
8: return true
```

method 2. Method 3 is represented by algorithms 5 and 6. Observe that the idea of method 3 is close to the idea of method 2. We present them both to show, in section 2.1.2, how experimental results can be misleading in analyzing the performance of method 2 versus method 3.

Analogous to algorithm 2, algorithm 5 traverses the defeaters. However, algorithm 5 checks if a defeater can be defeated by a member in the relevant set instead of delegating that task to another algorithm as it is in the case of algorithm 2. If a defeater is defeated, then algorithm 5 calls algorithm 6 to decide whether that defense is reinstated or not. The duty of algorithm 6 is two-fold. The first one is similar to the task of algorithm 3 while the other duty is in line with the job of algorithm 4. In algorithm 6, RED color designates that an attack is under processing and subsequently infinite procedure invocation is avoided.

Referring to the Modgil argument system in figure 2.1. To determine the acceptability of y to $S = \{x, y\}$, algorithm 5 finds (line 2) that $z \rightarrow^S y$. As $x \rightarrow^S z$ (line 4), algorithm 5 invokes algorithm 6 to check the reinstatement for $x \rightarrow^S z$. Now, algorithm 6 finds that $w \twoheadrightarrow (x \rightharpoonup z)$ (line 2) and finds that $y \rightarrow^S w$ (line 4). At this stage, algorithm 6 calls itself to determine the reinstatement for $y \rightarrow^S w$ (line 8). Subsequently, algorithm 6 finds that $z \twoheadrightarrow (y \rightharpoonup w)$ (line 2) and $x \rightarrow^S z$ (line 4). Since $x \rightarrow^S z$ is RED (line 5), this indicates that the reinstatement is assured for both $y \rightarrow^S w$ and $x \rightarrow^S z$. Hence, y is acceptable w.r.t. S. Now, we introduce the proof of algorithms 5 and 6.

Algorithm 5: Acceptable($x \in A, S \subseteq A$)

```
1: for all (y,x) ∈ R do
2:    if y →^S x then
3:       for all z ∈ S do
4:          if z →^S y ∧ Reinstated((z,y),S) = true then
5:             go to line 1 for next iteration
6:       return false
7: return true
```

Proposition 2. *Let (A, R, D) be a Modgil argument system, $S \subseteq A$, $x \in A$ and $(y, x) \in R$. Then:*

1. *x is acceptable w.r.t. S if and only if algorithm 5 returns true.*

Algorithm 6: Reinstated($(y,x) \in R, S \subset A$)

```
1:  color (y,x) RED
2:  for all (z,(y,x)) ∈ D do
3:     for all w ∈ S do
4:        if w →^S z then
5:           if (w,z) is RED then
6:              go to line 2 for next iteration
7:           else
8:              if Reinstated((w,z),S) then
9:                 go to line 2 for next iteration
10:    color (y,x) GREEN
11:    return false
12: color (y,x) GREEN
13: return true
```

2. *x is not acceptable w.r.t. S if and only if algorithm 5 returns false.*

3. *Let $y \rightarrow^S x$, then there is a reinstatement set for $y \rightarrow^S x$ if and only if algorithm 6 returns true.*

4. *Let $y \rightarrow^S x$, then there is no reinstatement set for $y \rightarrow^S x$ if and only if algorithm 6 returns false.*

Proof:

1. If x is acceptable w.r.t. S then for every $y \rightarrow^S x$ there is $z \in S : z \rightarrow^S y$ and there is a reinstatement set for $z \rightarrow^S y$ (definition 5). Algorithm 5 returns true (line 7) if for every $y \rightarrow^S x$ (line 2), there is $z \in S : z \rightarrow^S y$ and there is a reinstatement set for $z \rightarrow^S y$ (line 4).

2. If x is not acceptable w.r.t. S then there is $y \rightarrow^S x$ s.t. for each $z \in S$, $z \nrightarrow^S y$ or there is no reinstatement set for $z \rightarrow^S y$ (definition 5). Algorithm 5 returns false (line 6) if there exists $y \rightarrow^S x$ (line 2) s.t. for each $z \in S$, $z \nrightarrow^S y$ or there is no reinstatement set for $z \rightarrow^S y$ (line 4).

3. Algorithm 6 returns true (line 13) if for every $z \twoheadrightarrow (y \rightharpoonup x)$ (line 2), there is $w \in S : w \rightarrow^S z$ (line 4) and there is a reinstatement set for $w \rightarrow^S z$ (line 8).

4. Algorithm 6 returns false (line 11) if there is $z \twoheadrightarrow (y \rightharpoonup x)$ (line 2) s.t. for every $w \in S$, $w \nrightarrow^S z$ (line 4) or there is no reinstatement set for $w \rightarrow^S z$ (line 8).

■

As to which method is the fastest in practice, it is far from clear. In the subsequent section we present the experiments that evaluate these three methods.

2.1.2 Experiments

We start with a description of the conducted experiments then we show how to use the results of these experiments in analyzing the performance of the methods discussed in section 2.1.1. We implemented the experiments using Java on a Fedora (release 13) based machine[3] of 4 processors (Intel core i5-750 2.67GHz) and 16GB of memory. Instances of Modgil's argument system were generated randomly and approximately with equal probability. In particular, if $y \in A$ would attack two arguments and two attacks (note that deciding the number of attacks originated from a given argument can be random as well). Then $x \in A \setminus \{y\}$ is selected to be attacked by y with a probability of $\frac{1}{|A|-1}$, and subsequently, $z \in A \setminus \{y, x\}$ is selected to be attacked by y with a probability of $\frac{1}{|A|-2}$. Likewise, $(t, u) \in R$ is selected to be attacked by y with a probability of $\frac{1}{|R|}$ and then $(v, w) \in R \setminus \{(t, u)\}$ is selected to be attacked by y with a probability of $\frac{1}{|R|-1}$. In testing the implemented algorithms, we ran more than 100,000 Modgil argument systems. In the experiments we consider two criteria to measure the behavior of the methods. The first criterion is the average elapsed time in milliseconds denoted as α_{time}. We used the Java method *System.currentTimeMillis()* to measure the elapsed time. The second comparison criterion is the average total of elements of D processed in an execution, we denote this criteria by α_d. Note that the second criterion is implementation independent. In general, we cannot rely solely on the elapsed time in evaluating different algorithms because such a measure depends on the implementation of the algorithms (i.e. programming/coding) as well as the underlying structure of these algorithms.

Thus, we conducted three experiments for three questions:

1. which method is the most efficient as $|A|$ grows?
2. what is the impact of the increase in $|R|$ on the performance of the methods?
3. how do the methods behave as $|D|$ grows?

In all experiments we pictured the behavior of the methods in solving the credulous/skeptical acceptance problems. As to the first experiment, we generated 1000 Modgil argument systems where $|A|$ ranges from 6 to 20, the number of attacks is limited up to 7 per argument and the number of attacks on attacks is limited up to 4 per argument. The behavior of all methods is captured by figures 2.2, 2.3, 2.4 and 2.5. In these figures we use logarithmic scale. In the second experiment, we generated 1000 Modgil argument systems where $|R|$ varies from 12, 24, 36 ... to 132, $|A|$ as 12

[3]This is the machine for all experiments conducted in this work.

and the number of attacks on attacks is 2 per argument. Figures 2.6 and 2.7 trace the results of this experiment. In the third experiment, we generated 1000 Modgil argument systems where $|D|$ varies from 12, 24, 36 ... to 132, $|A|$ as 12 and the number of attacks on arguments is 4 per argument. The outcome of this experiment is shown in figures 2.8 and 2.9. To ensure accuracy, we repeated these experiments on different data with various parameters other than those mentioned above and the trends were completely consistent. Among several versions, we present here only one version of the conducted experiments. Moreover, observe that by these experiments we mean to know which algorithm is the fastest and hence there is no need, although it is possible, to consider very large argument systems as long as there are sufficient clues about algorithms' behavior under systems not considered to be very large. The size of instances might be of concern if the main objective is to study the underlying problem behavior, in that case it might be important to see how the computations are behaving in very large argument systems as well as small ones.

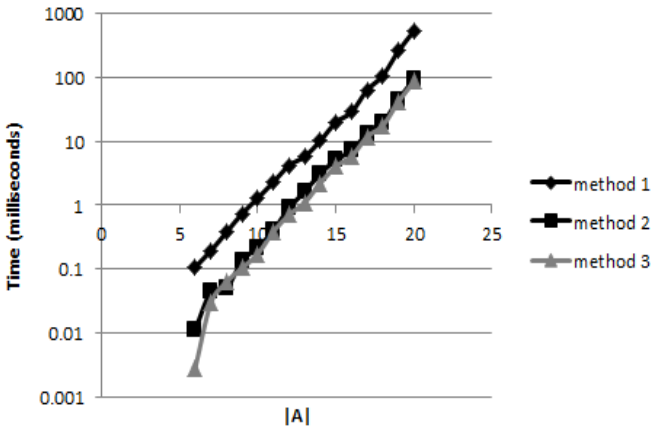

Figure 2.2: Deciding credulous acceptance, trend of α_{time} as $|A|$ grows.

To analyze, figures 2.2, 2.3, 2.4 and 2.5 show the performance for all methods indicating that methods 2 and 3 are faster than method 1 in deciding credulous/skeptical acceptance. However, although figures 2.2 and 2.4 suggest that methods 2 and 3 have nearly similar α_{time} trends, figures 2.3 and 2.5 indicate that method 2 is more efficient in terms of α_d. In such situation where two measures do not agree we seek a third one to decide which method is the most efficient. The overhead of procedure invocation might resolve the conflict, bearing in mind that method 2 is composed of 3 procedures and method 3 comprises two only. Figures 2.10 and 2.11 indicate that method 2 has a higher number of procedure invocations than method 3, and conclusively, methods 2 and 3 perform comparably. By examining the effect of $|R|$, figures 2.6 and 2.7 suggest that method 2 runs faster as $|R|$ grows in deciding credulous/skeptical acceptance. Figures 2.8 and 2.9 indicate that the rise in $|D|$ almost has an inverse impact on the effi-

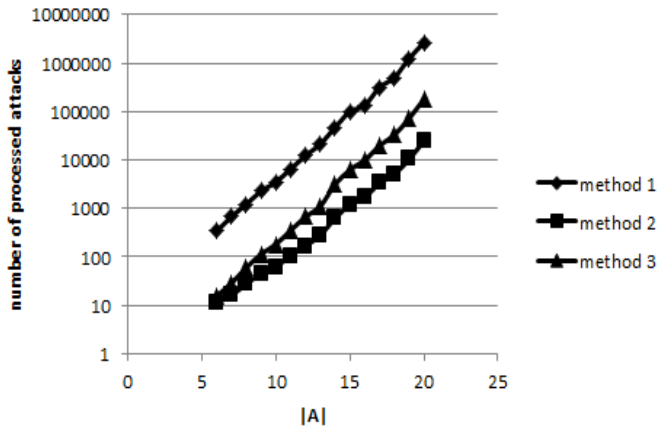

Figure 2.3: Deciding credulous acceptance, trend of α_d as $|A|$ grows.

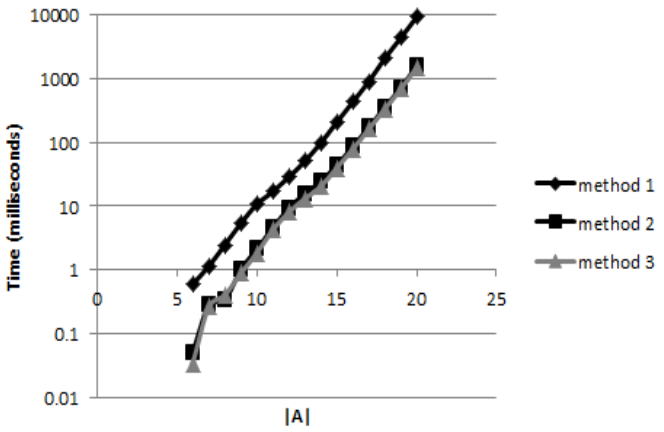

Figure 2.4: Deciding skeptical acceptance, trend of α_{time} as $|A|$ grows.

ciency of method 2 in solving the credulous/skeptical acceptance problems. Although figures 2.6, 2.7, 2.8 and 2.9 depict the behavior of method 2, the other methods showed similar trends, which are omitted to avoid redundancy. Data tables for all charts are presented in appendix A.

2.2 Experimental Algorithms for Value Based Argument Systems

For preliminaries of value based argument systems we refer the reader to section 1.5. Deciding Objective/subjective acceptance is believed to be intractable [40, 68]. However, as in any intractable problem, there are classes of value based argument systems that could be solved in linear time. The following proposition identifies value based argument systems that allow for linear time reasoning. In particular, problem instances with unrestricted number of arguments sharing the same value are solvable trivially

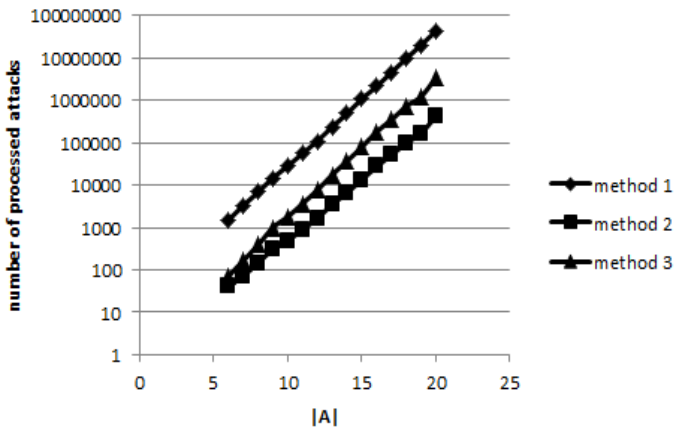

Figure 2.5: Deciding skeptical acceptance, trend of α_d as $|A|$ grows.

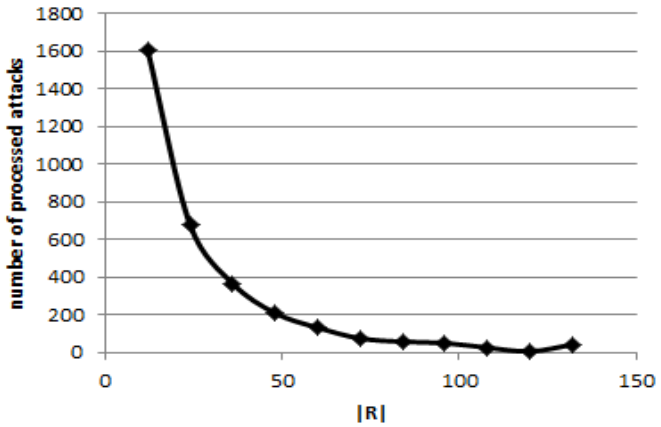

Figure 2.6: Trend of α_d as $|R|$ grows in method 2 deciding credulous acceptance.

with only one property: none of the attacks involves arguments sharing the same value.

Proposition 3. *Let* (A, R, V, η) *be a value based argument system. Then unattacked arguments are objectively accepted while the attacked ones are subjectively accepted if* $\forall (x, y) \in R\ (\eta(x) \neq \eta(y))$.

Proof: It is straightforward that unattacked arguments are objectively accepted. By contradiction we can prove that the remaining attacked arguments are subjectively accepted. Assume that not all of the attacked arguments are subjectively accepted, then one or more are either indefensible (i.e. neither objectively nor subjectively accepted) or objectively accepted. By definition an objectively accepted argument is attacked only by indefensible arguments. In fact, an indefensible argument x is attacked by accepted argument(s) that promote(s) $\eta(x)$. Contradiction. ∎

A naive approach to deciding objective/subjective acceptance needs to check all

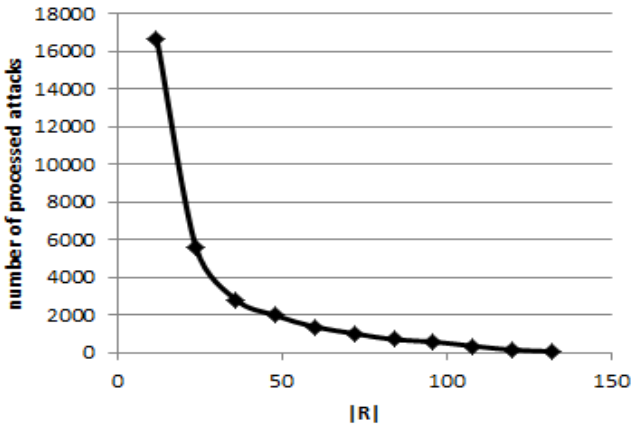

Figure 2.7: Trend of α_d as $|R|$ grows in method 2 deciding skeptical acceptance.

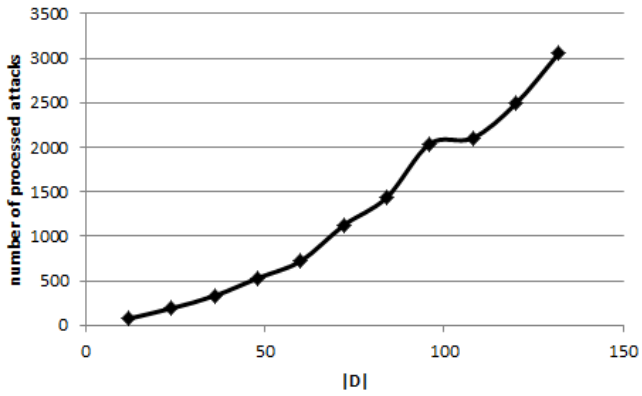

Figure 2.8: Trend of α_d as $|D|$ grows in method 2 deciding credulous acceptance.

total value orders (i.e. permutations of values in V). Thus, the acceptance of an argument will be computed in time proportional to $|V|!$ even if the underlying value based argument system is solvable in linear time. This stimulates us to look for a different approach that checks the minimum required value orders to decide acceptance, and so, in the next subsection we end up with algorithms having an improved running time.

2.2.1 Algorithms for Deciding Arguments' Acceptance

The idea of the new algorithm is based on the notion that acceptance of arguments is decided according to the acceptance of attackers. This notion has been already employed in algorithmic aspects of argument systems (see e.g. [78]). Indeed, the new algorithm searches the tree induced by a given argument s.t. the argument is the root and the children are the attackers and the children of these are their attackers and so on, provided that values are not repeated in a single directed path unless the repetition

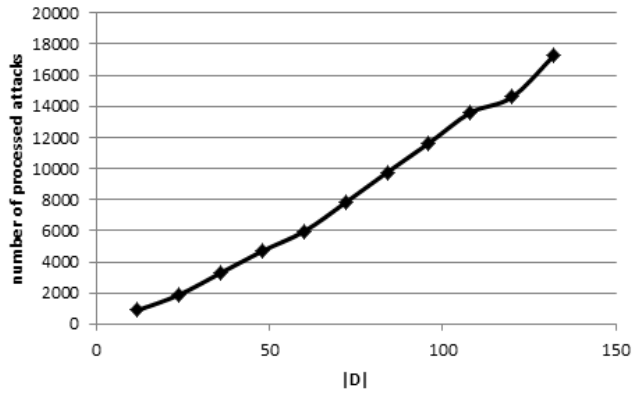

Figure 2.9: Trend of α_d as $|D|$ grows in method 2 deciding skeptical acceptance.

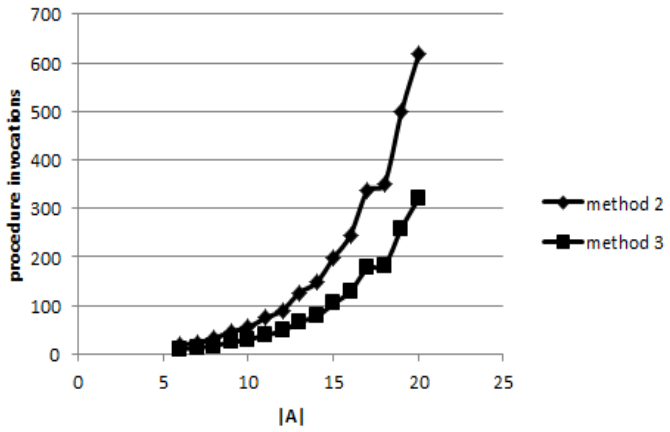

Figure 2.10: Trend of average of procedure invocations as $|A|$ grows in deciding credulous acceptance.

happens in a row. We call such a tree the *dispute tree*.

Definition 7. *Let (A, R, V, η) be a value based argument system. Then the dispute tree induced by $x \in A$, denoted by T_x, has x as the root and $\forall y, z \in A$, y is a child of z if and only if $(y,z) \in R$ and ($\eta(y) = \eta(z)$ or $\eta(y)$ does not appear on the directed path from z to x).*

Example 1. *Consider the value based argument system in figure 2.12 (throughout the dissertation we use argument-value as labels for nodes). The dispute tree T_y of y is depicted in figure 2.13. Note that we do not consider attacks with repeated values unless they are consecutive, for instance, in T_y the attack from x against z is dropped since x has the same value of y. The dropped attack is unsuccessful if v_2 is preferred to v_1, and it is unreachable if v_1 is preferred to v_2.*

Actually the new approach works on dispute trees. Before presenting the formal algorithms it might be helpful to discuss an example to capture the general idea.

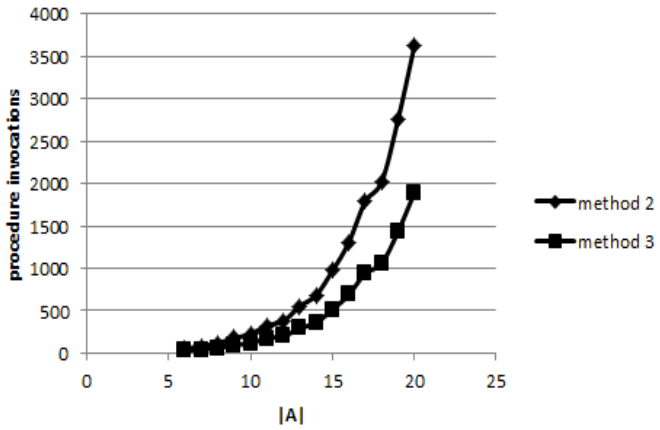

Figure 2.11: Trend of average of procedure invocations as $|A|$ grows in deciding skeptical acceptance.

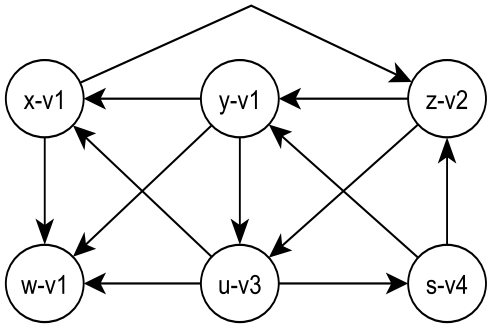

Figure 2.12: A value based argument system.

Example 2. *Consider the value based argument system in figure 2.12. The new approach decides that y is subjectively accepted since it is not defeated if v_1 is most preferred (see figure 2.14) and defeated when v_2 is most preferred (see figure 2.15). In total, y is subjectively accepted.*

Every time a social value is considered most preferred, the dispute tree changes accordingly. We refer to the new resulting tree as the *pruned dispute tree* under the superiority of some value.

Definition 8. *Let (A, R, V, η) be a value based argument system, $x \in A$, $v \in V$ and $T = T_x$. Then the pruned dispute tree under the superiority of v, denoted by T_v, is defined as $\{(y,z) \in T \mid \eta(y) = \eta(z) \vee \eta(z) \neq v\}$.*

Example 3. *Let T be the tree T_y in figure 2.13. Then $T_{v_2} = \{(z,y),(s,y),(u,s),(z,u)\}$.*

One more helpful term we have to define is related to the recursive nature of the new approach. Recall that we decide the acceptance of an argument based on

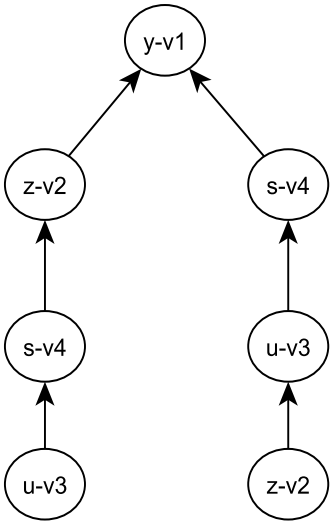

Figure 2.13: Referring to figure 2.12, the dispute tree T_y of y.

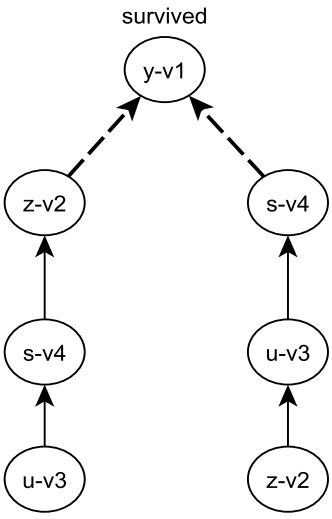

Figure 2.14: Referring to figure 2.12, the dispute tree of y if v_1 is most preferred noting that dashed-arrows represent dropped attacks.

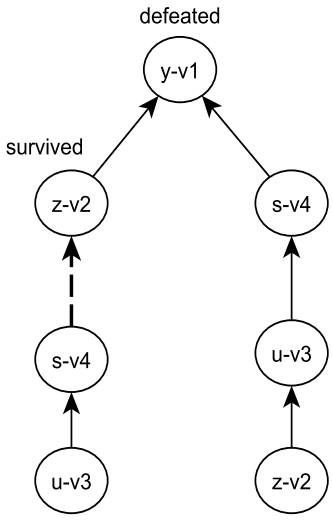

Figure 2.15: Referring to figure 2.12, the dispute tree of y if v_2 is most preferred noting that dashed-arrows represent dropped attacks.

attackers' acceptance, and thus, the acceptance of an attacker is computed according to the dispute subtree that is branched from that attacker.

Definition 9. *Let (A, R, V, η) be a value based argument system, $x \in A$ and $T = T_x$. Then the subtree $T(y)$ is defined as $\{(z,s) \in T \mid s = y \vee$ there is a directed path in T from s to $y\}$.*

Example 4. *Let T be the tree T_{v_2} from example 3. Then $T(s) = \{(u,s),(z,u)\}$.*

The new approach to objective/subjective acceptance is formally presented in algorithms 7 and 8. Algorithm 7 decides $status_x$ that denotes the objective/subjective acceptance of $x \in A$ while algorithm 8 decides $status'_x$ that denotes whether or not x is in a preferred extension under the superiority of some social value. Note that the second parameter, T, of algorithm 7 is initially, i.e. in the first invocation, equal to T_x where x is the argument in question.

Example 5. *Consider the value based argument system in figure 2.12, to find the acceptance of w the values $\{v_1, v_2, v_3, v_4\}$ are to be investigated. Now, if v_1 is most preferred then w is defeated as depicted in figure 2.16. Note that dispute trees in all figures are constructed left-to-right while arguments' statuses are decided bottom-up. Back to the acceptance of w, in the same way w is defeated if v_2 and v_3 are most preferred respectively (see figures 2.17 & 2.18). However, if v_4 is most preferred then w is undecided (see figure 2.19). At this stage, two more value orders are to be explored, namely, v_1 is preferred to v_3 and v_3 is preferred to v_1. In fact, w is also defeated in the latter two cases (see figures 2.20 and 2.21), and therefore, w is indefensible. To appreciate the benefit of algorithms 7 and 8, observe that a naive method would need to check 24 value orders (i.e. $|V|!$) to decide the acceptance of w, but the developed algorithms find the acceptance after checking only 6 value orders.*

Algorithm 7: DecideStatus($x \in A, T \subseteq T_x$)

for all $v \in V : \exists (y,z) \in T$ *s.t.* $(\eta(y) = v \vee \eta(z) = v)$ **do**
  $status'_x \leftarrow StatusAtValue(x, T, v)$
  if ($status_x$ *is* 2 *or null*) $\wedge\, status'_x = 2$ **then**
    $status_x \leftarrow 2$
  else
    if ($status_x$ *is* 0 *or null*) $\wedge\, status'_x = 0$ **then**
      $status_x \leftarrow 0$
    else
      if $status'_x = 1 \ \wedge \ T$ is not chain **then**
        $s \leftarrow DecideStatus(x, T_v)$
        if ($status_x$ *is* 2 *or null*) $\wedge\, s = 2$ **then**
          $status_x \leftarrow 2$
        else
          if ($status_x$ *is* 0 *or null*) $\wedge\, s = 0$ **then**
            $status_x \leftarrow 0$
          else
            return 1
      else
        return 1
return $status_x$

Algorithm 8: StatusAtValue($x \in A, T \subseteq T_x, v \in V$)

$status'_x \leftarrow 2$
for all $y \in A$ *s.t.* $(y,x) \in T$ **do**
  if $\eta(x) \neq v \vee \eta(y) = \eta(x)$ **then**
    $status'_y = StatusAtValue(y, T(y), v)$
    if $status'_y = 2 \wedge (\eta(x) = \eta(y) \vee \eta(y) = v)$ **then**
      return 0
    else
      if $status'_y \neq 0$ **then**
        $status'_x = 1$
return $status'_x$

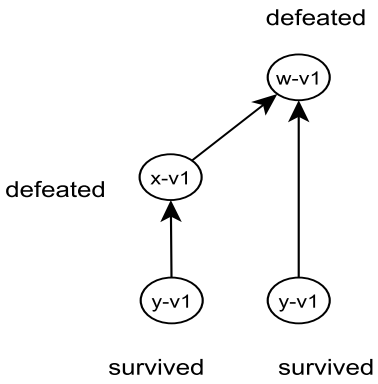

Figure 2.16: Progress of algorithms in example 5: the state when v_1 is most preferred.

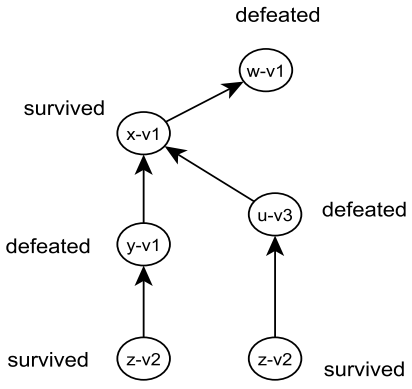

Figure 2.17: Progress of algorithms in example 5, the state when v_2 is most preferred.

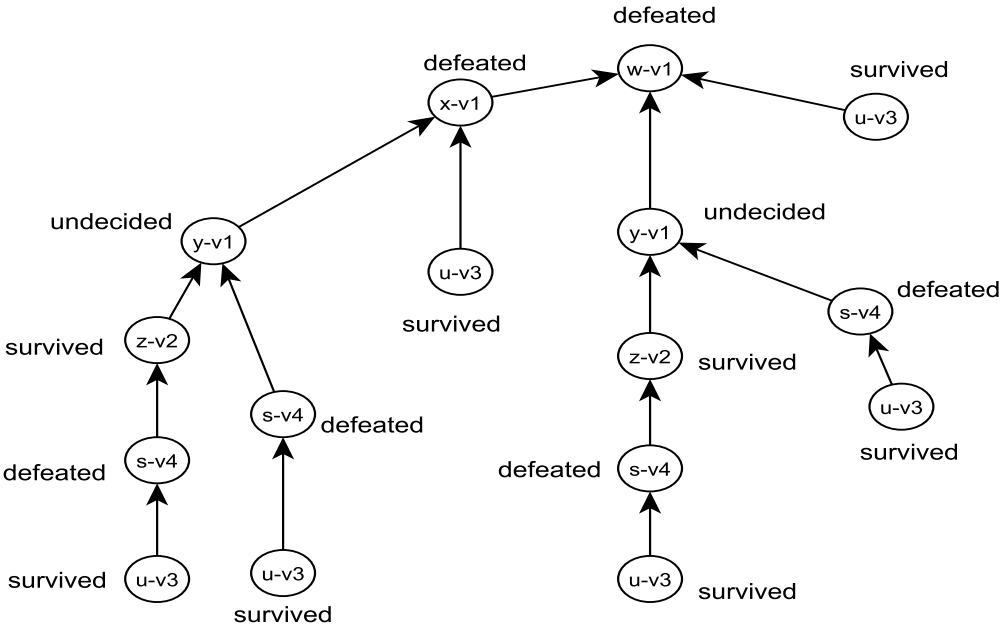

Figure 2.18: Progress of algorithms in example 5, the state when v_3 is most preferred.

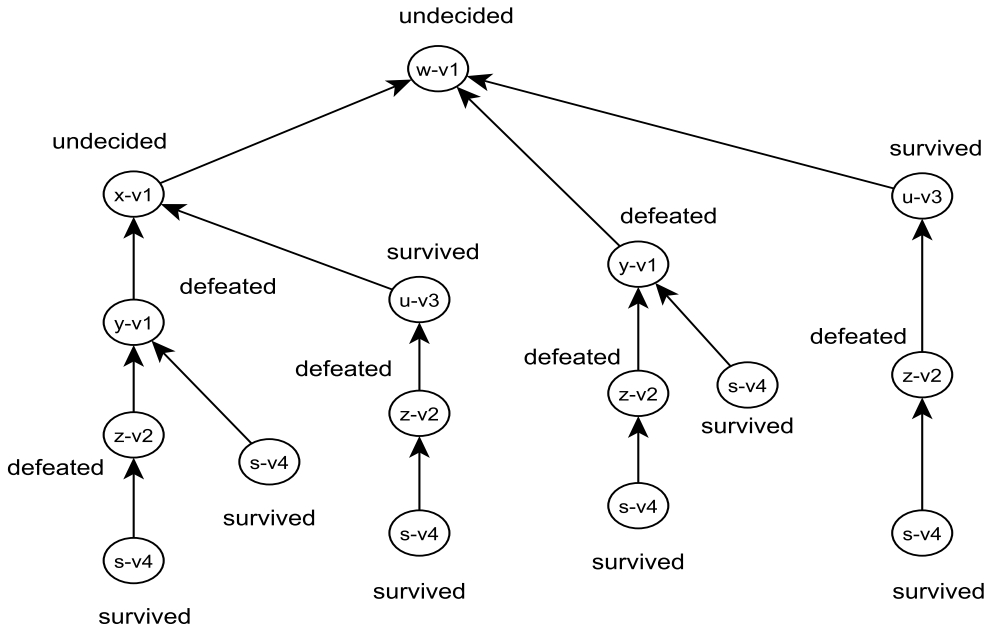

Figure 2.19: Progress of algorithms in example 5, the state when v_4 is most preferred.

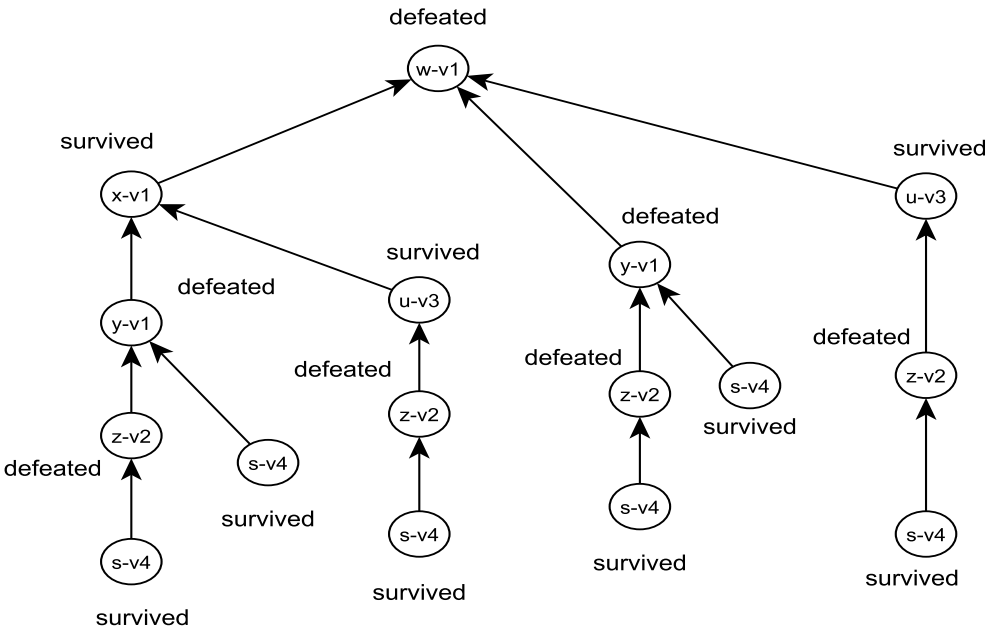

Figure 2.20: Progress of algorithms in example 5, the state when v_4 is most preferred and v_1 is preferred to v_3.

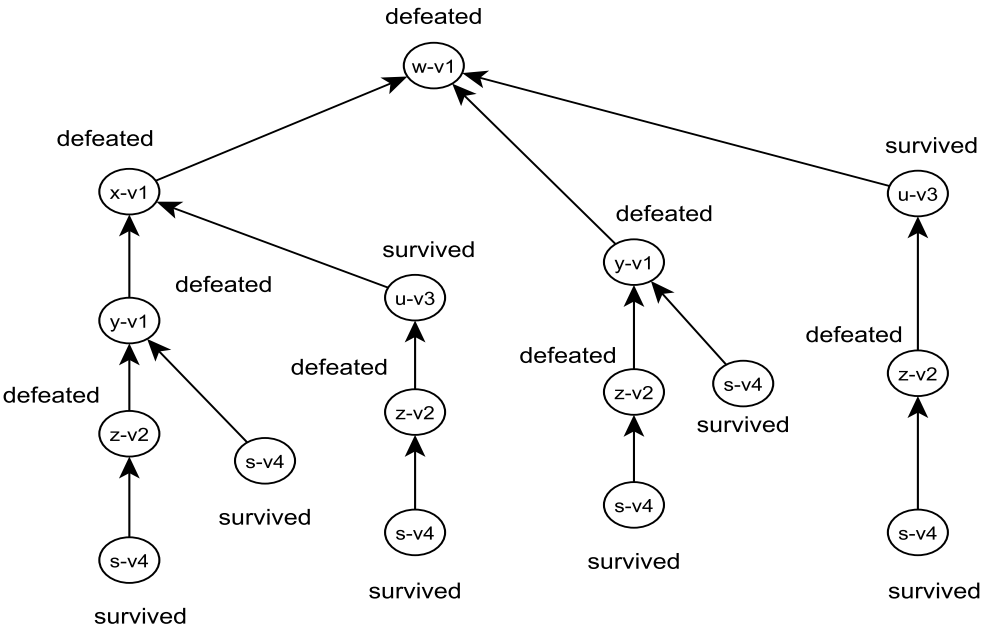

Figure 2.21: Progress of algorithms in example 5, the state when v_4 is most preferred and v_3 is preferred to v_1.

In the remainder of this section we present the proof of algorithms 7 and 8. The algorithms are recursive and so the proof is by induction. These proofs basically identify the base and inductive cases that are directly provable from the definition of subjective/objective acceptance and the other definitions presented in this section. In fact, algorithm 8 is a depth first search procedure that visits the attackers of the argument in question, say x, until it returns 0 indicating that x is not a member of a preferred extension, 1 indicating undecided (w.r.t. whether or not x in a preferred extension) or 2 indicating that x is in a preferred extension.

Proposition 4. *Let (A, R, V, η) be a value based argument system and $x \in A$. Under the superiority of $v \in V$ over any other social value promoted by arguments in the tree T, algorithm 8 returns $status'_x$ equal to:*

1. *0 if and only if $\forall S \subseteq A : S$ is admissible $(x \notin S)$.*
2. *2 if and only if $\exists S \subseteq A : S$ is admissible with $x \in S$.*
3. *1 otherwise.*

Proof: In the base case the algorithm stops and returns 2 for one of two reasons. Firstly, if there is no attacker against x at all (line 2). Secondly, if $\eta(x) = v$ and all attackers of x in T do not promote v (line 3). This follows directly from the defeat notion in value based argument systems. The inductive case: given y attacks x such that $status'_y = 2$, i.e. y is in an admissible set, and $\eta(y) = v$ or $\eta(y) = \eta(x)$ then $status'_x$ is 0 (line 6), otherwise, if $status'_y$ is not equal to 0 then $status'_x = 1$ (i.e. undecided, see line 9). The inductive case follows directly from the fact: with respect to any total value order there is only one preferred extension under the assumption of multi-value cycles [13]. The base case establishes (2) while the inductive case shows (1) & (3). ∎

Algorithm 7 is also a recursive procedure that returns: 0 for indefensible arguments, 1 for subjective acceptance or 2 for objective acceptance.

Proposition 5. *Let (A, R, V, η) be a value based argument system and $x \in A$. Then algorithm 7 returns $status_x$ equal to:*

1. *0 if and only if x is indefensible.*
2. *1 if and only if x is subjectively accepted.*
3. *2 if and only if x is objectively accepted.*

Proof: The proof follows directly from proposition 4 and the definition of subjective/objective acceptance. In the base case the algorithm stops for one of two reasons. Firstly, if there are no values left for investigation then the algorithm returns $status_x$ as 0 or 2 (lines 1, 4, 7, 12, 15 & 20). Secondly, the algorithm returns 1 if $status_x$ is not *null*

and $status_x \neq status'_x$ and ($status'_x \neq 1$ or T is a chain) (line 19). Inductive case: if $status_x$ is *null* or 0 (respectively 2) then $status_x$ stays 0 (respectively 2) if the status of x w.r.t. T_v is 0 (respectively 2), see lines 10-15. Otherwise, $status_x$ is 1 (line 17). ∎

2.2.2 Experiments

We implemented the algorithms presented in the previous section in Java on a Linux-based machine of 4 CPUs (Intel core i5-750 2.67GHz) and 16GB of memory. We generated instances of value based argument systems for these experiments such that all steps are implemented randomly with approximately equal probability. These steps include choosing the number of arguments, number of social values and number of attacks, which arguments attack which others and finally which argument is mapped to which value. We tested the algorithm with 100,000 value based argument systems where $|V|$ ranges from 2 to 7 and $|A|$ ranges from 2 to 15.

We ran three experiments. For each experiment we aimed at generating 10000 value based argument systems. But then we had to drop inapplicable instances, which have a cycle of arguments promoting the same social value. The first experiment was to show how our algorithm compares to a naive approach, in which every total value order is examined in order to decide subjective/objective acceptance. For this purpose, we experimented 9844 value based argument systems grouped by $|V|$. Table 2.1 details each group while table 2.2 presents the average total of value orders processed in an execution, denoted as $\alpha_{value-order}$, and the average elapsed time in milliseconds, denoted as α_{time}, for each group under the new algorithm versus a naive algorithm. We used the Java method *System.currentTimeMillis()* to track the elapsed time. The second experiment was to show how our algorithm's behavior is affected by the increase of $|V|$. Figures 2.22 and 2.23 show the behavior in terms $\alpha_{value-order}$ and α_{time} respectively. The charts in figures 2.22 and 2.23 are obtained from 9753 value based argument systems where the number of attacks against any argument is limited up to 4, $|A|$ is 30 and $|V|$ ranges from 2 to 20. The last experiment was to evaluate, in the context of the new algorithm, the correlation between the number of attacks against any single argument and the performance measured by $\alpha_{value-order}$ and α_{time}. The results of the last experiment are presented in figures 2.24 and 2.25 where the charts plot 9500 value based argument systems with $|A|$ as 20, $|V|$ as 4 and the number of attacks against any argument ranges from 2 to 20. As an illustration on how to read these figures, the point (15,83.22) in figure 2.22 means that for a collection of value based argument systems with $|V| = 15$ the algorithm needs to check on average 83.22 value orders to decide the acceptance. The data tables for all figures are presented in appendix A.

To sum up, the outcome of the experiments shows that the new algorithm has on average a better behavior than a naive algorithm as stated by table 2.2. Figures 2.22 and 2.23 point that the time complexity of the new algorithm might be exponential,

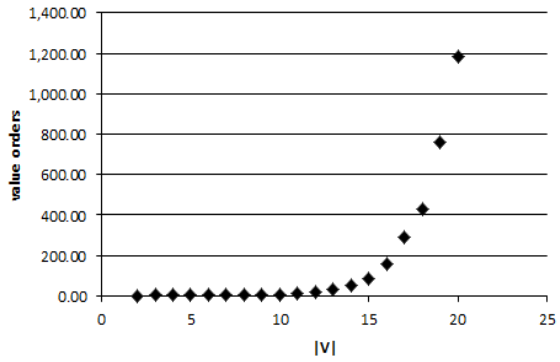

Figure 2.22: The effect of increase in $|V|$.

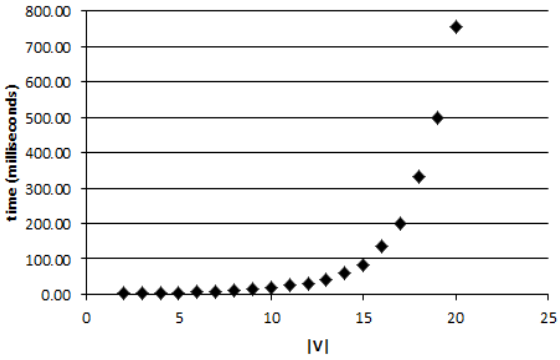

Figure 2.23: The effect of increase in $|V|$.

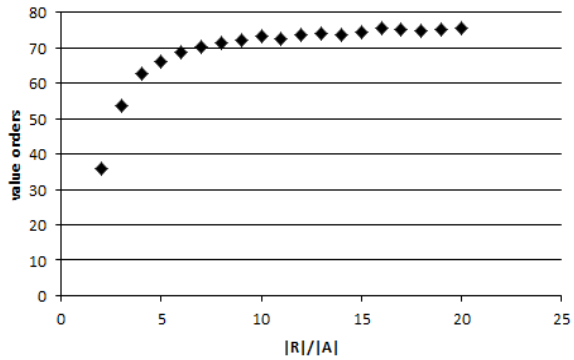

Figure 2.24: The effect of increase in attacks per argument.

Table 2.1: Value based argument systems referenced by table 2.2.

group	$\|V\|$	$\|$*value based argument systems*$\|$	*range*($\|A\|$)	*range*($\|R\|$)
1	2	1284	2-12	0-45
2	3	1400	3-15	0-58
3	4	1505	4-17	1-87
4	5	1422	5-18	3-927
5	6	1405	7-20	8-109
6	7	1404	8-20	10-135
7	8	996	8-20	13-140
8	9	428	9-20	20-138

Table 2.2: The new algorithm versus a naive algorithm.

group	the new algorithm		a naive algorithm	
	$\alpha_{value-order}$	α_{time}	$\alpha_{value-order}$	α_{time}
1	1.66	0.04	2.00	0.03
2	2.37	0.11	6.00	0.15
3	3.10	0.33	24.00	1.06
4	3.83	0.91	120.00	7.17
5	4.53	3.46	720.00	31.20
6	5.12	11.00	5,040.00	221.36
7	5.60	32.91	40,320.00	2,438.76
8	5.92	103.40	362,880.00	45,676.53

which is not surprising since the problem of subjective/objective acceptance is believed to be hard (see e.g. [40]). Finally, figures 2.24 and 2.25 show that the increase in the number of attacks against any single argument has no extreme impact on the behavior of the new algorithm.

2.3 Summary

We presented a case study on experimental algorithms in the context of two instances of extended argument systems. The concern was to show the potential of empirical analysis in evaluating algorithms for decision problems in argument systems. Analyzing algorithms is a crucial process in engineering practical algorithms. The role of experimental methodology becomes evident in cases where theoretical evaluation might fall short in discovering the performance differences between algorithms.

We projected our discussions on algorithms in value based argument systems and Modgil's argument system. This should not be construed as giving favor to these systems over other formalisms such as [1, 57]. We did not mean in this chapter to analyze the semantics of the concerned formalisms or even to show the link between them and other related argumentation theories. Although we examined the efficiency of

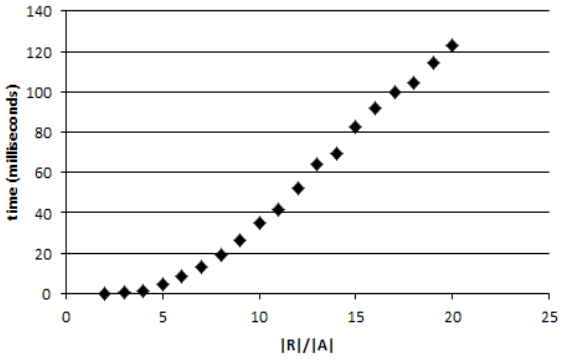

Figure 2.25: The effect of increase in attacks per argument.

somewhat simple algorithms, we believe that the experimental treatments introduced in this chapter are applicable to other algorithmic issues in argument systems.

Considering experiments as a means for evaluating the performance of algorithms is generally not new in the field of computer science (see e.g. [74]). However, making use of empirical investigations in the context of argument systems for the purpose of developing algorithms is, to the best of our knowledge, considered by only a few works. In [54] experiments were conducted to evaluate the effectiveness of an algorithm for constructing logic-based arguments. In [70] experiments have been conducted to examine approximation versus exact computations in the context of argument systems while [12] empirically evaluated the effect of splitting a given argument system on the computation of preferred extensions. We see these previous works as complementary to our experimental study presented in this chapter in the sense that experimental algorithmics give insights to various features according to the objectives of the research question under study. This does not imply, by no means, that experiments in the general sense have not been undertaken by other works for objectives away from designing and analyzing dedicated algorithms, see for example [53, 51, 19].

Empirical evaluation of algorithms can be taken further to several algorithmic issues in the context of arguments systems as we show in the coming chapters. For example, [28, 97, 95] have proposed algorithms to decide credulous/skeptical acceptance. From an application perspective it is still an open question concerning which of these algorithms is the most efficient in practice.

Chapter 3

New Algorithms for Preferred Semantics

Doutre and Mengin [35] and later Modgil and Caminada [78] presented algorithms for computing preferred extensions. Informally, the two algorithms are based on the notion that arguments which might be included in an extension are labeled *IN* while arguments which might not be in the respective extension are labeled *OUT* and the undecided arguments are labeled *UNDEC*. Both algorithms start with some initial label for all arguments and then the labels change, through what are so-called *transitions*, several times until some condition holds. At this point, the arguments labeled IN make up an admissible set. These algorithms go through different sequences of transitions, and hence, they find admissible sets in order to construct the preferred extensions. Nonetheless, the two algorithms mainly differ in two issues. Firstly, the arguments' initial labels. Secondly, the transitions of arguments' labels. As we show, these issues affect the performance significantly.

The contribution of this chapter can be summarized in four points. Firstly, we improve labels' transitions by utilizing further labels, and hence, the preferred extensions are enumerated faster than existing algorithms. Secondly, we introduce a new mechanism for pruning the search space such that transitions leading to "dead ends" are avoided at an early stage. Thirdly, we present a cost-effective heuristic rule that selects arguments for transitions such that a goal state (i.e. a preferred extension) might be achieved earlier. Fourthly, by incorporating the three improvements, we design algorithms for answering the skeptical/credulous acceptance question without explicitly enumerating all preferred extensions.

We develop in section 3.1 a new algorithm that enumerates all preferred extensions. Supported by experiments presented in section 3.3, we argue in section 3.2 that our algorithm is faster in deciding the preferred extensions than the existing algorithms of Doutre and Mengin [35] and Modgil and Caminada [78]. Regarding the acceptance problem, the skeptically/credulously accepted arguments might be simply decided by enumerating all preferred extensions. However, in situations where the acceptance

problem is confined to a specific argument then it is more efficient to avoid enumerating all preferred extensions explicitly, especially when the underlying argument system is dynamic (i.e. changes frequently such as in a dialog setting). Adhering to this view, in section 3.4 we engineer algorithms for the acceptance problem that outperform, with respect to running time, the existing algorithms of Cayrol et al. [28], Thang et al. [95] and the algorithm of Verheij for the credulous acceptance problem [97]. With respect to the acceptance algorithms, we introduce comparisons with existing algorithms and empirical evaluation in section 3.5. Lastly, we offer further discussions, review of related works and conclusions in section 3.6.

3.1 Preferred Extension Enumeration: The New Algorithm

Besides applying the labels IN, OUT and UNDEC that are used by the existing algorithms of [35, 78], we introduce the use of MUST_OUT and IGNORED. In what follows we informally explain the usage of the five labels in our algorithm. The IN label identifies arguments that might be in a preferred extension. The OUT label identifies an argument that is attacked by an IN argument. The UNDEC label is for any unprocessed argument whose final label is not decided yet. The MUST_OUT label identifies arguments that attack IN arguments. The IGNORED label designates arguments which might not be included in a preferred extension because they might not be defended by any IN argument. The precise usage of these labels is introduced shortly.

To enumerate the preferred extensions our algorithm starts with UNDEC as the initial label for all arguments. Next, the algorithm forks to two sets of new labels via two kinds of transitions: *IN-TRANS* and *IGNORE-TRANS*. During IN-TRANS three actions are taken. Firstly, an UNDEC argument becomes IN. Secondly, attackers of the newly IN argument become MUST_OUT. Thirdly, arguments attacked by the newly IN argument are labeled OUT.

Definition 10. *Let (A, R) be an argument system, $x \in A$, $Lab : A \rightarrow \{IN, OUT, MUST\_OUT, IGNORED, UNDEC\}$ be a total mapping such that $Lab(x) = UNDEC$. Then the in transition on x, denoted as IN-TRANS(x), is defined by the next ordered actions:*

1. *$Lab' \leftarrow Lab$, and then*
2. *$Lab'(x) \leftarrow IN$, and then*
3. ***for all** $(x, y) \in R$ **do** $Lab'(y) \leftarrow OUT$, and then*
4. ***for all** $(z, x) \in R : Lab'(z) \neq OUT$ **do** $Lab'(z) \leftarrow MUST\_OUT$, and then*
5. ***return** Lab'.*

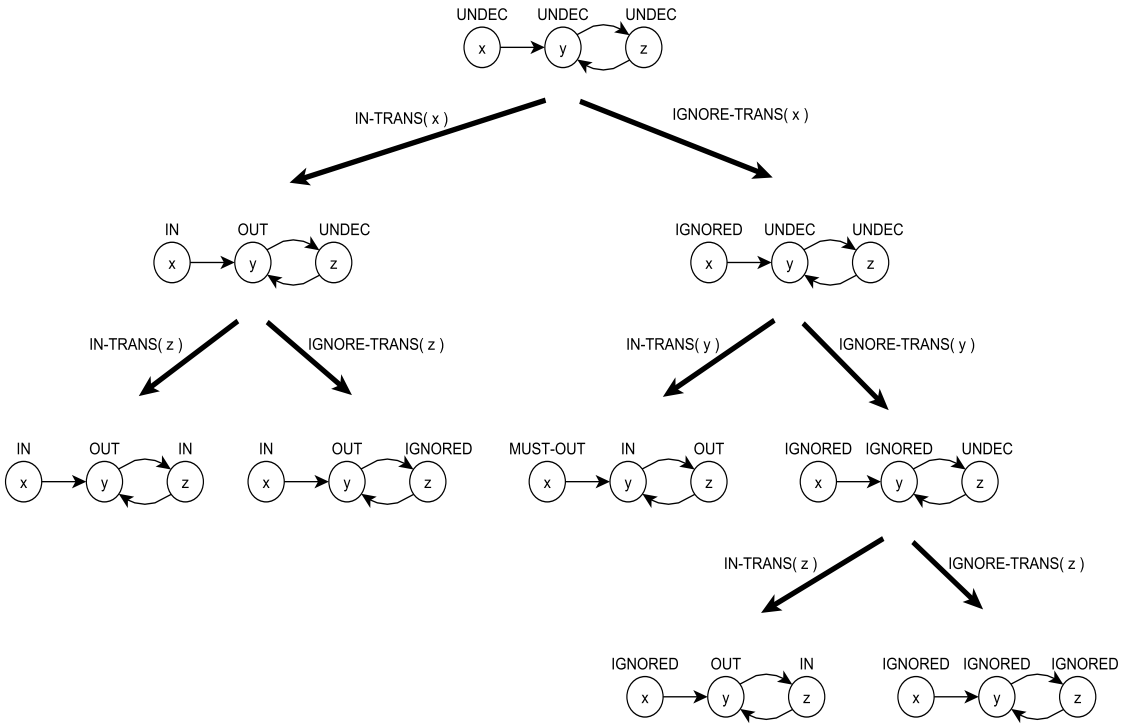

Figure 3.1: How algorithm 9 works on an argument system.

The purpose of the IGNORE-TRANS is to try to find a preferred extension excluding the IGNORED argument. Hence, the IGNORE-TRANS is applied by labeling an argument IGNORED as the following definition states.

Definition 11. *Let* (A,R) *be an argument system,* $x \in A$ *,* $Lab : A \rightarrow \{IN, OUT, MUST\_OUT, IGNORED, UNDEC\}$ *be a total mapping such that* $Lab(x) = UNDEC$*. Then the ignore transition on x, denoted as IGNORE-TRANS(x), is defined by the following steps:*

1. $Lab' \leftarrow Lab$*, and then*
2. $Lab'(x) \leftarrow IGNORED$*, and then*
3. ***return*** Lab'*.*

Algorithm 9 defines formally our approach. The idea of algorithm 9 is to change arguments' labels via IN-TRANS and IGNORE-TRANS repeatedly until there is no argument that is UNDEC. At this stage, the IN arguments make up an admissible set if and only if no argument is MUST_OUT. Figure 3.1 shows how algorithm 9 works on the argument system of figure 1.2.

Let us now improve the efficiency of algorithm 9 by applying three enhancements. For the first enhancement, algorithm 9 selects an UNDEC argument for IN-TRANS arbitrarily (line 12); however, as we demonstrate it is more productive to apply the following selection rule:

Algorithm 9: Enumerating all preferred extensions of an argument system (A, R).

```
1  Lab : A → {IN, OUT, MUST_OUT, IGNORED, UNDEC}; Lab ← φ;
2  foreach x ∈ A do Lab ← Lab ∪ {(x, UNDEC)};
3  E_preferred ⊆ 2^A; E_preferred ← φ;
4  call find-preferred-extensions(Lab);
5  report E_preferred is the set of all preferred extensions;

6  procedure find-preferred-extensions(Lab)
7  if ∄x ∈ A : Lab(x) = UNDEC then
8      if ∄x ∈ A : Lab(x) = MUST_OUT then
9          S ← {y ∈ A | Lab(y) = IN};
10         if ∄T ∈ E_preferred : S ⊆ T then E_preferred ← E_preferred ∪ {S};
11 else
12     select any x ∈ A s.t. Lab(x) = UNDEC;
13     Lab' ← IN-TRANS(x);
14     call find-preferred-extensions(Lab');
15     Lab' ← IGNORE-TRANS(x);
16     call find-preferred-extensions(Lab');
17 end procedure
```

1. select any $x \in A$ s.t. x is UNDEC and for each $(y, x) \in R$, y is OUT or MUST_OUT.

2. otherwise select any $x \in A$ s.t. x is UNDEC and $|\{x\}^+|$ is maximal.

Later in this section we will explain the reason behind the first part of this selection rule. As to the second part, the intuition is that this might accelerate reaching a goal state, that is, an admissible set. Recall that the set of IN arguments is admissible if and only if all arguments in the system are IN, OUT or IGNORED. Thus, the goal state(s) of the search might be reached faster as much as we minimize the number of UNDEC/MUST_OUT arguments by maximizing the number of OUT arguments[1]. In the opposite way, as long as the first part of the selection rule failed, one might pick up an UNDEC argument x for IN-TRANS such that the number of arguments that attack x is minimal. At first sight, such selection seems sensible because it produces almost a minimal number of MUST_OUT arguments. However, recall that we get to a goal state (i.e. admissible set) if and only if no argument is UNDEC or MUST_OUT, and thus, minimizing the number of MUST_OUT arguments will not be helpful as far as the number of UNDEC arguments is not also minimized. In section 3.3 we show experimentally the efficiency of our heuristic rule against some other options.

For the second enhancement to algorithm 9, we reactivate a pruning mechanism (originally used by [35] but here we improve the utilization to maximize the profit as we explain in section 3.2.1) that detects the branch of the search space that will eventually take us to a dead end. In particular, the pruning mechanism says that if

[1]This is also noted by [33] in deciding stable extensions.

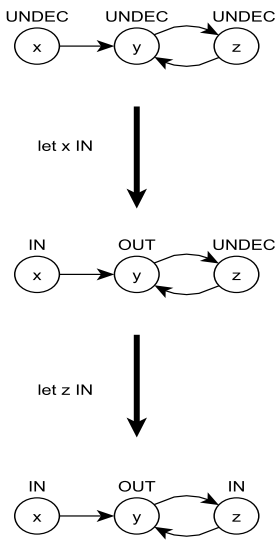

Figure 3.2: How algorithm 10 works on an example argument system.

at any state of the search there exists a MUST_OUT argument that is not attacked by an UNDEC argument then proceeding further is fruitless and so the algorithm simply must backtrack.

For the third enhancement, we use a further pruning tactic which skips ignoring an argument x that is attacked only by OUT or MUST_OUT arguments; note that if an admissible set S will be found while such x is ignored then $S \cup \{x\}$ is certainly admissible and so there is no need to ignore x in the first place. At this point it is convenient to show the grounds of the first part of our heuristics that selects an UNDEC argument x such that for each $(y, x) \in R$ y is OUT or MUST_OUT; notice that the earlier we make such x IN, the bigger part of the search tree that would be bypassed (keeping in mind the third enhancement).

Furthermore, we modify algorithm 9 by two minor changes. For the first change, note that IGNORE-TRANS does only change the transitioned argument, and so, there is no need to fork to a new set of labels via IGNORE-TRANS. Simply, we drop the IGNORE-TRANS and label the transitioned argument IGNORED. For the second change, we rewrite the IN-TRANS by its definition to make the algorithm self-contained. Now, we give algorithm 10 that reinforces algorithm 9 by incorporating the three enhancements mentioned before alongside the two minor changes. Figure 3.2 shows how algorithm 10 computes the preferred extensions of the argument system of figure 1.2. To prove algorithm 10 we have to show three issues. Firstly, the IN arguments make up an admissible set if and only if no argument is UNDEC or MUST_OUT. Secondly, a decided admissible set is a preferred extension if and only if the set is not a subset of any previously decided preferred extension. Thirdly, algorithm 10 discovers all preferred extensions.

Algorithm 10: Improvement of algorithm 9 that enumerates preferred extensions of an argument system (A, R).

```
1  N(y) ≡ |{z ∈ A | (y,z) ∈ R}|;
2  Lab : A → {IN, OUT, MUST_OUT, IGNORED, UNDEC}; Lab ← φ;
3  foreach x ∈ A do Lab ← Lab ∪ {(x, UNDEC)};
4  E_preferred ⊆ 2^A; E_preferred ← φ;
5  call find-preferred-extensions(Lab);
6  report E_preferred is the set of all preferred extensions;

7  procedure find-preferred-extensions(Lab)
8  while ∃y ∈ A : Lab(y) = UNDEC do
9      select y ∈ A s.t. Lab(y) = UNDEC and ∀(z,y) ∈ R
       (Lab(z) ∈ {OUT, MUST_OUT}), otherwise select y ∈ A s.t. Lab(y) = UNDEC
       and ∀z ∈ A : Lab(z) = UNDEC, N(y) ≥ N(z);
10     Lab' ← Lab;
11     Lab'(y) ← IN;
12     foreach (y,z) ∈ R do Lab'(z) ← OUT;
13     foreach (z,y) ∈ R do
14         if Lab'(z) ∈ {IGNORED, UNDEC} then
15             Lab'(z) ← MUST_OUT;
16             if ∄(w,z) ∈ R : Lab'(w) = UNDEC then
17                 Lab(y) ← IGNORED;
18                 goto line 8;
19     call find-preferred-extensions(Lab');
20     if ∃(z,y) ∈ R : Lab(z) ∈ {UNDEC, IGNORED} then
21         Lab(y) ← IGNORED;
22     else
23         Lab ← Lab';
24 if ∄y ∈ A : Lab(y) = MUST_OUT then
25     S ← {y ∈ A | Lab(y) = IN};
26     if ∄T ∈ E_preferred : S ⊆ T then E_preferred ← E_preferred ∪ {S};
27 end procedure
```

Proposition 6. *Let* (A, R) *be an argument system and* $E_{preferred}$ *be the set of subsets of* A *reported by algorithm 10. Then:*

1. $\forall S \in E_{preferred}, \exists T \subseteq A : T$ *is a preferred extension* $\wedge\ T = S$.
2. $\forall T \subseteq A$, *if* T *is a preferred extension then* $\exists S \in E_{preferred} : S = T$.

Proof: Firstly, we demonstrate that for each $S \in E_{preferred}$, S is admissible. To establish that S is conflict free, assume that there is $z, y \in S$ *s.t.* $(z, y) \in R$. Thus, y would be labeled OUT according to algorithm 10, see line 12. This contradicts line 25, which implies that for each $x \in S$ $Lab(x) = IN$. To show that for each $x \in S$ x is acceptable w.r.t. S, suppose that there is $y \in S$, $(z, y) \in R$ with no $w \in S$ *s.t.* $(w, z) \in R$. So, z would be labeled MUST_OUT according to lines 13-15. This contradicts the actions of lines 24-26, which basically report S admissible and subsequently add S to $E_{preferred}$ if and only if there does not exist $w \in A$ *s.t.* $Lab(w) = MUST\_OUT$.

Secondly we need to prove the maximality (w.r.t. $\subseteq$) of the elements in $E_{preferred}$. Assume that there is $S \in E_{preferred}$ *s.t.* S is not maximal. Given the action of line 26, there exists an admissible set $SS \supset S$ s.t. $SS \notin E_{preferred}$. This basically means there is $y \in SS$ *s.t.* $y \notin S$. This contradicts the actions of algorithm 10 by which it firstly labels y IN (see line 11) and then later IGNORED (see lines 17 and 21). These actions basically imply that SS would be discovered before S and consequently SS would be added to $E_{preferred}$ according to line 26.

Lastly, it follows directly that algorithm 10 finds all preferred extensions. Note that algorithm 10 examines all subsets of A by labeling every argument y initially labeled UNDEC to be IN (see line 11) and afterwards IGNORED (see lines 17 and 21) which reflects the exploration of all subsets that include, respectively exclude, y. ∎

3.2 The Advantage of the New Algorithm over Existing Algorithms

3.2.1 The Algorithm of Doutre and Mengin

In [35] Doutre and Mengin (DM for short) present an algorithm to enumerate all preferred extensions using three labels: IN, OUT and UNDEC. As in algorithm 10 that we have presented, DM algorithm starts with all arguments UNDEC and then the algorithm forks via two transitions iteratively. However, there are five differences between DM algorithm and algorithm 10 as illustrated in the following.

Firstly, DM algorithm selects an UNDEC argument for transitions according to somewhat expensive heuristic rules such that if one rule fails to select an argument another rule is applied and so forth. Here we give three DM rules:

R1. DM selects an UNDEC argument x s.t. the following conditions hold altogether:

   i. $\forall(y,x)$, y is not IN nor UNDEC and y is attacked by x or an IN argument,

   ii. $\forall(x,z)$, z is not IN.

R2. DM selects an UNDEC argument x s.t. for each $(y,x) \in R$, y is not attacked by an IN (or UNDEC) argument.

R3. DM selects an UNDEC argument x s.t. there exists $z \in A$ *s.t.* z is IN, $(y,z) \in R$, $(x,y) \in R$ and for each $(w,y) \in R$ *s.t.* $w \neq x$, w is OUT.

By comparing only these rules (leaving aside the other DM rules) against our rules (which, we recall, select an UNDEC argument x s.t. for each $(y,x) \in R$, y is OUT or MUST_OUT otherwise the rule selects an UNDEC argument x such that $|\{x\}^+|$ is maximal), one can see that our heuristic rules are potentially computationally lighter than the DM rules. Furthermore, we present in section 3.3 experiments indicating that our rules are more cost-effective than the heuristics of DM.

Secondly, in DM algorithm the counterpart transition of IN-TRANS labels the attackers of an IN argument OUT while in our approach such attackers are labeled MUST_OUT. The benefit of the MUST_OUT label is to streamline a pruning mechanism as we demonstrate next. DM algorithm stops exploring a branch further and backtracks if there is $(x,y) \in R$ *s.t.* x is OUT, y is IN and for each $(z,x) \in R$ z is OUT; this is checked after *every* transition. Nevertheless, algorithm 10 backtracks if a MUST_OUT argument is not attacked by an UNDEC argument; this is checked only during the IN-TRANS that produces new MUST_OUT arguments. We note that algorithm 10 needs to check the condition of this pruning strategy less frequently than the algorithm of DM. Besides, this pruning check is verified on average more efficiently in algorithm 10 compared to DM algorithm. Notice that searching for a MUST_OUT argument runs in the order of $|A|$ while searching for an OUT argument that attacks an IN argument runs in the order of $|R|$; typically $|R| > |A|$.

Thirdly, the DM counterpart of IGNORE-TRANS labels the respective argument as OUT instead of IGNORED. To appreciate the benefit of the IGNORED label consider what follows. Once the DM algorithm finds an admissible set, the labels of all arguments are either IN or OUT, and thus, one cannot tell which of the OUT arguments are in conflict with the IN arguments. Comparing with our algorithm, an admissible set is reported if and only if all arguments are IN, OUT or IGNORED, and in effect, one easily can see that the OUT arguments attack (or are attacked by) an IN argument while the IGNORED arguments are those which are excluded from the respective admissible set since they might be indefensible by the IN arguments.

Fourthly, to ensure the maximality of the reported preferred extensions DM algorithm checks that for each $T \subseteq \{y : y \text{ is } OUT\}$, $T \cup \{x : x \text{ is } IN\}$ is not admissible. This

is more expensive than the approach of algorithm 10 that simply reports a maximal admissible set S if and only if S is not a subset of a previously decided preferred extension.

Fifthly, we stress that our algorithm utilizes a new pruning mechanism that skips ignoring an argument that is attacked only by OUT or MUST_OUT arguments as we illustrated earlier in section 3.1.

3.2.2 The Algorithm of Modgil and Caminada

In [78] Modgil and Caminada (MC) present an algorithm to enumerate all preferred extensions using three labels: IN, OUT and UNDEC. We show six differences between MC algorithm and algorithm 10.

Firstly, MC approach starts with all arguments IN while our approach starts with all arguments UNDEC. We believe that this is not a stylistic issue, the initial state dramatically affects the performance, keeping in mind that the initial state defines what kind of transitions are applicable. Transitions bear two efficiency factors. Firstly, the computations incurred by applying the transition itself. Secondly, the number of applicable transitions by which the algorithm might fork at a time. In what follows we emphasize on these two issues.

Secondly, for transitions MC select an IN argument (*super-illegally* in MC terms) that is attacked by a legally IN (or UNDEC) argument. An IN argument x is legally IN if and only if for each $(y,x) \in R$, y is OUT. An UNDEC argument x is legally UNDEC if and only if x is not attacked by an IN argument while x is attacked by an UNDEC argument. If there are no super-illegally arguments then MC algorithm picks an illegally IN argument, i.e. an argument attacked by an IN (or UNDEC) argument. Contrastingly, algorithm 10 picks up an UNDEC argument x s.t. for each $(y,x) \in R$, y is OUT or MUST_OUT otherwise algorithm 10 selects an UNDEC argument x such that $|\{x\}^+|$ is maximal. Therefore, the selection process of MC runs in the order of $|R|^2$ while in algorithm 10 the selection process runs in the order of $|R|$.

Thirdly, in MC approach there is one kind of transition which is completely different from our transitions. In MC transition, an illegally IN argument is changed to OUT and due to this change any OUT argument (*illegally* OUT in MC terminology) that is not attacked by an IN argument anymore is changed to UNDEC. Hence, in MC the transitions need to process the attackers of a set of OUT arguments. This set contains the newly OUT argument plus the OUT arguments which are attacked by that newly OUT argument. We note that our transition methods are computationally cheaper than the transition method of MC: the IGNORE-TRANS only processes one argument while the IN-TRANS processes the attackers of the newly IN argument plus the arguments which are attacked by that newly IN argument.

Fourthly, MC algorithm might find, at any stage, several illegally IN arguments,

and so, each illegally IN argument induces a transition. Consider the case where every argument attacks all of the other arguments, then there are $|A|$ illegally IN arguments, and subsequently, there are $|A|$ transitions. Afterwards, there are $|A|-1$ illegally IN arguments and so forth. Thus, MC algorithm runs in the order of $|A|!$ transitions. However, algorithm 10 (DM algorithm as well) runs in the order of $2^{|A|}$ due to the fact that our algorithm forks via exactly two transitions at any time.

Fifthly, to ensure the maximality (w.r.t. $\subseteq$) of the decided preferred extensions MC algorithm drops, once there is no illegally IN argument, the decided admissible sets which are subsets of the current set of IN arguments. As we have shown, algorithm 10 reports a maximal admissible set if the set of IN arguments is not a subset of any previously decided preferred extension. Then, it is usually the case that algorithm 10 bears fewer computations to warrant the maximality since the number of preferred extensions is typically less than the number of admissible sets.

Sixthly, MC algorithm incorporates a pruning mechanism which is different from the one used in algorithm 10. MC algorithm stops and backtracks if the set of IN arguments is a subset of a previously decided admissible set. In essence experimental evaluation presented in section 3.3 suggests that our pruning mechanism is more powerful.

3.3 Empirical Evaluation

We undertook experiments to verify that algorithm 10 is more efficient than existing algorithms. In the next section we describe the settings of our experiments.

3.3.1 Experiments Description

All the algorithms of this thesis, new and previous ones, were implemented in C++[2] on a Fedora (release 13) based machine with 4 processors (Intel core i5-750 2.67GHz) and 16GB of memory. We tested the algorithms of this chapter with at least 100,000 randomly generated argument systems. We generated instances of argument systems by two methods. The first method is described by algorithm 11, while the other method is done by setting an attack between two arguments with a specific probability, which we declare for each reported experiment. As we show, the first method is more representative w.r.t. $|R|$, i.e. algorithm 11 generates small $|R|$ as well as somewhat large $|R|$, while generating argument systems with a specific probability will produce instances with nearly similar density w.r.t. $|R|$. Hence, we believe that using algorithm 11 is sufficient for the purpose of our experiments. However, we also consider generating argument systems by setting attacks with a specific probability because it is, perhaps, a more common method than algorithm 11, and therefore, we mean to remove any doubts

[2] Apart from the algorithms of chapter 2, which are implemented in Java.

that a skeptical reader might have. To compare between algorithms we tracked the average elapsed time in milliseconds, denoted by α_{time}. The elapsed time was obtained by using the `time` command of Linux. In addition, to provide a platform-independent measure, we reported the average total number of attacks processed during an execution, denoted by $\alpha_{attacks}$. Each measurement of α_{time} or $\alpha_{attacks}$ represents the average for 100 generated argument systems where each system might have a different $|R|$. This section describes our experimental settings in general for the rest of this thesis, and so, we might refer to this section wherever we report experimental results.

Algorithm 11: Generating an instance of an argument system (A, R)

$A \leftarrow \{a_1, a_2 ... a_n\}$;
$R \leftarrow \phi$;
$s \leftarrow$ Choose (uniformly at random) a natural number between 1 and $n-1$;
// i.e. $P[s = k] = 1/(n-1)$;
foreach $i : 1 \leq i \leq n$ **do**
    $t \leftarrow$ Choose (uniformly at random) a natural number between 0 and s;
    // i.e. $P[t = k] = 1/(s+1)$;
    foreach $k : 1 \leq k \leq t$ **do**
        $j \leftarrow$ Choose (uniformly at random) a natural number between 1 and n s.t. $j \neq i$ and $(a_i, a_j) \notin R$;
        $R \leftarrow R \cup \{(a_i, a_j)\}$;

3.3.2 Evaluating the Algorithm in Contrast to Existing Algorithms

As tables 3.1 and 3.2 indicate, algorithm 10 is more efficient than the algorithm of Doutre and Mengin [35]. Similarly, tables 3.3 and 3.4 suggest that algorithm 10 outperforms the algorithm of Modgil and Caminada [78]. Moreover, we compare algorithm 10 with dynPARTIX [52] which is based on a dynamic programming algorithm where a tree decomposition is computed for the given argument system and then the preferred extensions are decided by working on that decomposition, and thus, the time complexity of the algorithm mainly depends on the tree width of the given argument system while it is linear in the size of the argument system. As suggested by tables 3.5 and 3.6, algorithm 10 is faster than dynPARTIX (version 2).

3.3.3 Evaluating Heuristics

In selecting an UNDEC argument x for IN-TRANS, we consider three selection strategies in the event of there is no $x \in A$ *with* x is UNDEC satisfying for each $(y, x) \in R$, y is OUT or MUST_OUT. Specifically

$SELC_1$. select x such that $|\{x\}^+|$ is maximal.

$SELC_2$. select x uniformly at random from those labeled UNDEC.

Table 3.1: Algorithm of Doutre and Mengin versus algorithm 10, argument systems were generated by using algorithm 11.

$\|A\|$	Range of $\|R\|$	Algorithm of Doutre and Mengin		Algorithm 10	
		α_{time}	$\alpha_{attacks}$	α_{time}	$\alpha_{attacks}$
16	16-152	42.90	173,985.18	10.10	1,079.52
17	17-185	82.50	342,125.92	10.00	1,484.46
18	18-190	138.10	606,814.63	10.80	2,033.13
19	19-208	235.80	1,043,640.02	10.00	2,298.23
20	20-232	526.40	2,227,440.78	27.00	3,317.15
21	21-269	1,269.10	5,135,244.41	16.30	4,274.53
22	22-279	1,964.60	8,250,082.03	14.10	5,354.97
23	23-306	4,499.40	18,103,628.02	13.60	7,833.88

Table 3.2: Algorithm of Doutre and Mengin versus algorithm 10, argument systems were generated by setting attacks with a probability of $\frac{2 \times log_e |A|}{|A|}$.

$\|A\|$	Algorithm of Doutre and Mengin		Algorithm 10	
	α_{time}	$\alpha_{attacks}$	α_{time}	$\alpha_{attacks}$
16	40.90	162,669.60	0.30	509.94
17	64.30	278,827.90	0.00	622.34
18	140.50	620,731.02	0.00	770.78
19	217.00	956,478.06	0.00	862.77
20	452.60	1,989,006.80	0.00	1,103.40
21	1,013.50	4,202,136.39	0.10	1,360.33
22	1,955.40	8,167,079.65	0.00	1,636.15
23	3,381.80	14,010,377.81	0.00	1,973.35

Table 3.3: Algorithm of Modgil and Caminada versus algorithm 10, argument systems were generated by using algorithm 11.

$\|A\|$	Range of $\|R\|$	Algorithm of Modgil and Caminada		Algorithm 10	
		α_{time}	$\alpha_{attacks}$	α_{time}	$\alpha_{attacks}$
7	7-36	13.90	116,748.55	10.10	55.83
8	8-44	27.40	878,620.66	10.10	81.96
9	9-53	123.00	7,518,451.58	10.30	120.10
10	10-67	1,341.30	95,136,946.42	11.30	165.20
11	11-81	17,718.70	1,289,875,022.46	16.80	230.57
12	12-102	198,260.00	14,843,459,628.78	20.50	297.07

Table 3.4: Algorithm of Modgil and Caminada versus algorithm 10, argument systems were generated by setting attacks with a probability of $\frac{2\times log_e|A|}{|A|}$.

$\|A\|$	Algorithm of Modgil and Caminada		Algorithm 10	
	α_{time}	$\alpha_{attacks}$	α_{time}	$\alpha_{attacks}$
7	10.40	180,048.15	0.00	65.89
8	47.20	2,468,480.49	0.00	100.31
9	245.50	16,156,341.51	0.00	124.11
10	1,717.40	122,267,084.12	0.00	149.15
11	13,839.10	1,022,422,791.17	0.00	176.05
12	129,694.00	9,398,228,010.93	2.80	222.20

Table 3.5: The average elapsed time of algorithm 10 versus dynPARTIX, argument systems were generated by using algorithm 11.

$\|A\|$	Range of $\|R\|$	dynPARTIX	Algorithm 10
21	21-255	61.70	36.00
22	22-268	91.70	37.50
23	23-299	148.80	38.00
24	24-337	298.90	30.00
25	25-332	223.80	33.33
26	26-369	485.30	40.00
27	27-458	650.40	35.00
28	28-470	1,431.70	30.00
29	29-467	1,525.50	26.00
30	30-484	2,933.40	25.00

Table 3.6: The average elapsed time of algorithm 10 versus dynPARTIX, argument systems were generated by setting attacks with a probability of $\frac{2\times log_e|A|}{|A|}$.

$\|A\|$	dynPARTIX	Algorithm 10
21	36.30	0.00
22	51.70	0.00
23	71.00	0.00
24	96.90	0.00
25	142.50	0.20
26	191.80	0.10
27	294.40	0.00
28	388.20	0.00
29	585.60	0.10
30	831.00	0.20

Table 3.7: Evaluating heuristics for algorithm 10 by tracing the average elapsed time, argument systems are generated by using algorithm 11.

$\|A\|$	Range of $\|R\|$	α_{time}		
		$SELC_1$	$SELC_2$	$SELC_3$
51	51-1420	33.10	79.90	109.60
52	52-1514	36.60	81.60	151.80
53	53-1565	70.10	156.70	276.10
54	54-1654	40.20	140.70	190.70
55	55-1542	50.40	118.00	162.80
56	56-1740	47.00	110.20	200.40
57	57-1747	99.70	198.80	395.30
58	58-1820	84.40	223.10	269.10
59	59-1919	104.30	275.80	428.60
60	60-2008	83.30	364.30	722.90

Table 3.8: Evaluating heuristics for algorithm 10 by tracing the average total attacks processed in an execution, argument systems were generated by using algorithm 11.

$\|A\|$	Range of $\|R\|$	$\alpha_{attacks}$		
		$SELC_1$	$SELC_2$	$SELC_3$
51	51-1420	56,674,350.00	419,600,980.00	497,036,320.00
52	52-1514	77,158,060.00	506,418,890.00	718,256,140.00
53	53-1565	142,804,040.00	885,072,160.00	1,260,449,180.00
54	54-1654	88,628,520.00	892,263,610.00	977,251,560.00
55	55-1542	98,386,710.00	708,626,820.00	777,678,830.00
56	56-1740	95,596,010.00	646,236,060.00	1,003,608,330.00
57	57-1747	218,721,530.00	1,061,332,270.00	1,700,097,880.00
58	58-1820	168,321,570.00	1,269,828,260.00	1,369,440,260.00
59	59-1919	202,274,190.00	1,521,032,860.00	1,882,353,340.00
60	60-2008	181,076,010.00	2,175,114,920.00	3,070,755,730.00

$SELC_3$. select x such that $|\{x\}^-|$ is minimal.

We evaluated all these selection options; tables 3.7, 3.8, 3.9 and 3.10 show that the first strategy is the most efficient. Recall that we illustrated the reasons behind this earlier in section 3.1.

3.4 Deciding Skeptical and Credulous Acceptance: New Algorithms

In deciding acceptance, it might be desirable to produce some kind of proof (i.e. explanation) as to why an argument is credulously accepted. In order to define what makes up a proof for the credulous acceptance let us recall a helpful term. We say that an argument x is *reachable* from an argument y if and only if there is a directed path from y to x. For example, consider the argument system depicted in figure 3.3 where

Table 3.9: Evaluating heuristics for algorithm 10 by tracing the average elapsed time, argument systems were generated by setting attacks with a probability of $\frac{2\times log_e|A|}{|A|}$.

$\|A\|$	α_{time}		
	$SELC_1$	$SELC_2$	$SELC_3$
51	23.10	44.40	60.00
52	28.60	51.20	75.40
53	34.20	61.30	96.50
54	39.30	66.90	111.70
55	35.30	61.90	90.70
56	38.40	72.10	108.70
57	43.60	84.70	122.70
58	53.40	101.40	153.40
59	59.50	114.50	183.60
60	71.40	141.70	216.10

Table 3.10: Evaluating heuristics for algorithm 10 by tracing the average total attacks processed in an execution, argument systems were generated by setting attacks with a probability of $\frac{2\times log_e|A|}{|A|}$.

$\|A\|$	$\alpha_{attacks}$		
	$SELC_1$	$SELC_2$	$SELC_3$
51	124,211,520.00	388,755,410.00	436,034,330.00
52	155,233,160.00	457,403,240.00	554,937,770.00
53	187,574,380.00	566,546,030.00	690,943,580.00
54	217,752,000.00	621,320,020.00	807,769,740.00
55	210,179,920.00	585,514,800.00	684,125,670.00
56	233,738,620.00	696,231,930.00	827,515,780.00
57	268,330,940.00	809,601,160.00	915,755,090.00
58	325,295,100.00	979,667,610.00	1,134,830,850.00
59	362,701,330.00	1,119,429,760.00	1,348,104,360.00
60	435,559,000.00	1,353,460,680.00	1,608,847,260.00

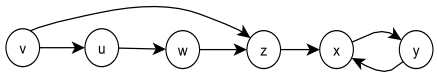

Figure 3.3: An argument system.

$A = \{u,v,w,x,y,z\}$ and $R = \{(u,w),(v,u),(w,z),(v,z),(z,x),(x,y),(y,x)\}$. In figure 3.3 x is reachable from u through the directed path $\langle(u,w),(w,z),(z,x)\rangle$ while u is not reachable from x. Thus, a *credulous proof* of a given argument is made up of two sets: an admissible set containing the argument and the set of all counter arguments.

Definition 12. *Let (A,R) be an argument system, $S \subseteq A$ be an admissible set containing x s.t. $\forall z \in S$ x is reachable from z. Then, $S \cup S^{-}$ is a credulous proof for x.*

It follows directly that our definition of the credulous proof is compatible with the definition of credulous acceptance. Note that a given argument is credulously accepted if and only if the argument is in an admissible set, which is explicitly expressed in definition 12. Algorithm 12 determines a credulous proof of an argument (should such exist).

Basically, algorithm 12 is a modification of algorithm 10 such that instead of finding all preferred extensions algorithm 12 tries to find an admissible set containing the argument in question. Hence, algorithm 12 makes use of six labels: *PRO* (short for proponent), *OPP* (short for opponent), *IGNORED*, *OUT*, *MUST_OUT* and UNDEC. An argument x is labeled PRO to indicate that x might be in an admissible set and the argument in question is reachable from x. An argument y is labeled OUT if and only if y is attacked by a PRO argument. The MUST_OUT label identifies arguments that attack PRO arguments. An argument y is labeled OPP if and only if y is attacked by a PRO argument and y attacks a PRO argument. An argument y is labeled IGNORED to signal that y cannot be in an admissible set with the current PRO arguments. The UNDEC label is for any unprocessed argument whose label is not decided yet. The precise usage of these labels is defined in algorithm 12. The basic notion of algorithm 12 is to change arguments' labels iteratively according to the labeling scheme outlined earlier until there does not exist an argument that is MUST_OUT. At this point, PRO/OPP arguments make up a credulous proof for the argument in question such that PRO arguments represent the admissible part of the proof.

Referring to the argument system in figure 3.3, $\{v,x,y,z\}$ is a credulous proof for x where $\{v,x\}$ is admissible, see figure 3.4 that demonstrates how algorithm 12 works. Although figure 3.4 does not reflect every aspect of algorithm 12, the figure might help the reader to capture the general idea. To prove algorithm 12, it is essential to show that PRO arguments make up an admissible set.

Proposition 7. *Let (A,R) be an argument system and $x \in A$. Then:*

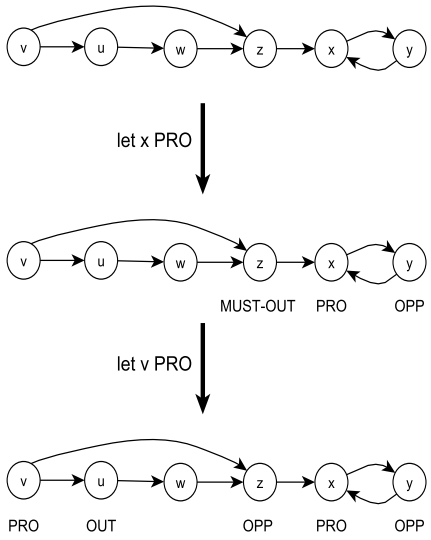

Figure 3.4: Deciding a credulous proof for the argument x by using algorithm 12.

1. *If algorithm 12 decides that x is credulously proved by* $T = \{y \in A \mid Lab(y) \in \{PRO, OPP\}\}$ *then* $\exists S \subseteq A : S$ *is admissible* $\wedge\ S = T \setminus \{y \in A \mid Lab(y) = OPP\}$.

2. *If x is credulously accepted then algorithm 12 decides that x is credulously proved by* $\{y \in A \mid Lab(y) \in \{PRO, OPP\}\}$.

Proof: To prove both parts, we need to show that $\{y \in A \mid Lab(y) = PRO\}$, denoted by SS, is admissible. To establish that SS is conflict free, assume that there are $z,\ y \in SS\ s.t.\ (z, y) \in R$. Thus, y would be labeled OUT or OPP according to algorithm 12, see lines 5, 12, 22, 25 and 34. This contradicts $SS = \{y \in A \mid Lab(y) = PRO\}$. To show that for each $y \in SS$ y is acceptable w.r.t. SS, suppose that there is $y \in SS$, $(z, y) \in R$ with no $w \in SS\ s.t.\ (w, z) \in R$. Thus, z would be labeled MUST_OUT according to lines 8 and 28. This contradicts the actions of lines 16 and 41. These actions imply that SS is reported admissible conditional on that there is no $w \in A$ *with* $Lab(w) = MUST\_OUT$. ∎

Regarding the decision problem of skeptical acceptance, the proof for a skeptically accepted argument x can be fulfilled by any admissible set containing x provided that there does not exist a preferred extension that does not contain x. We modified algorithm 10 to algorithm 13 that decides the skeptical acceptance of an argument x. Firstly, algorithm 13 looks for a credulously accepted argument that attacks x. If there exists such an attacker then algorithm 13 concludes that x is not skeptically accepted. Otherwise, algorithm 13 searches for a preferred extension that expels x. If such an extension is found then x is not skeptically accepted, or else x is skeptically accepted provided that x is in an admissible set S, and subsequently, S might form the skeptical proof of x. Algorithm 13 is somewhat self-explanatory. However, see figure 3.5 that shows how the algorithm works in deciding the skeptically accepted argument w in the

Algorithm 12: Constructing a credulous proof of an argument x in an argument system (A, R).

1 $N(y) \equiv |\{z \in A \mid (y,z) \in R\}|$;
2 $Lab : A \rightarrow \{PRO, OPP, OUT, MUST\_OUT, IGNORED, UNDEC\}$; $Lab \leftarrow \phi$;
3 **foreach** $y \in A$ **do** $Lab \leftarrow Lab \cup \{(y, UNDEC)\}$;
4 $Lab(x) \leftarrow PRO$;
5 **foreach** $(x,y) \in R$ **do** $Lab(y) \leftarrow OUT$;
6 **foreach** $(z,x) \in R$ **do**
7 **if** $Lab(z) = UNDEC$ **then**
8 $Lab(z) \leftarrow MUST\_OUT$;
9 **if** $\nexists (w,z) \in R : Lab(w) = UNDEC$ **then**
10 x is not credulously accepted; **exit**;
11 **else**
12 **if** $Lab(z) = OUT$ **then** $Lab(z) \leftarrow OPP$;
13 **if** is-accepted(Lab) = true **then** x is proved by $\{y \in A \mid Lab(y) \in \{PRO, OPP\}\}$;
14 **else** x is not credulously acceptable;

15 **procedure** is-accepted(Lab)
16 **foreach** $y \in A : Lab(y) = MUST\_OUT$ **do**
17 **while** $\exists (z,y) \in R : Lab(z) = UNDEC$ **do**
18 select $z \in A$ s.t. $Lab(z) = UNDEC$ and $(z,y) \in R$ and $\forall (w,z) \in R$ $(Lab(w) \in \{OUT, MUST\_OUT, OPP\})$, otherwise select $z \in A$ s.t. $Lab(z) = UNDEC$ and $(z,y) \in R$ and $\forall (w,y) \in R : Lab(w) = UNDEC,\ N(z) \geq N(w)$;
19 $Lab' \leftarrow Lab$; $Lab'(z) \leftarrow PRO$;
20 **foreach** $(z,u) \in R$ **do**
21 **if** $Lab'(u) = MUST\_OUT$ **then**
22 $Lab'(u) \leftarrow OPP$;
23 **else**
24 **if** $Lab'(u) \neq OPP$ **then**
25 $Lab'(u) \leftarrow OUT$;
26 **foreach** $(v,z) \in R$ **do**
27 **if** $Lab'(v) \in \{IGNORED, UNDEC\}$ **then**
28 $Lab'(v) \leftarrow MUST\_OUT$;
29 **if** $\nexists (w,v) \in R : Lab'(w) = UNDEC$ **then**
30 $Lab(z) \leftarrow IGNORED$;
31 **goto** line 17;
32 **else**
33 **if** $Lab'(v) = OUT$ **then**
34 $Lab'(v) \leftarrow OPP$;
35 **if** *is-accepted(Lab') = true* **then**
36 $Lab \leftarrow Lab'$;
37 return true;
38 **else**
39 $Lab(z) \leftarrow IGNORED$;
40 return false;
41 return true;
42 **end procedure**

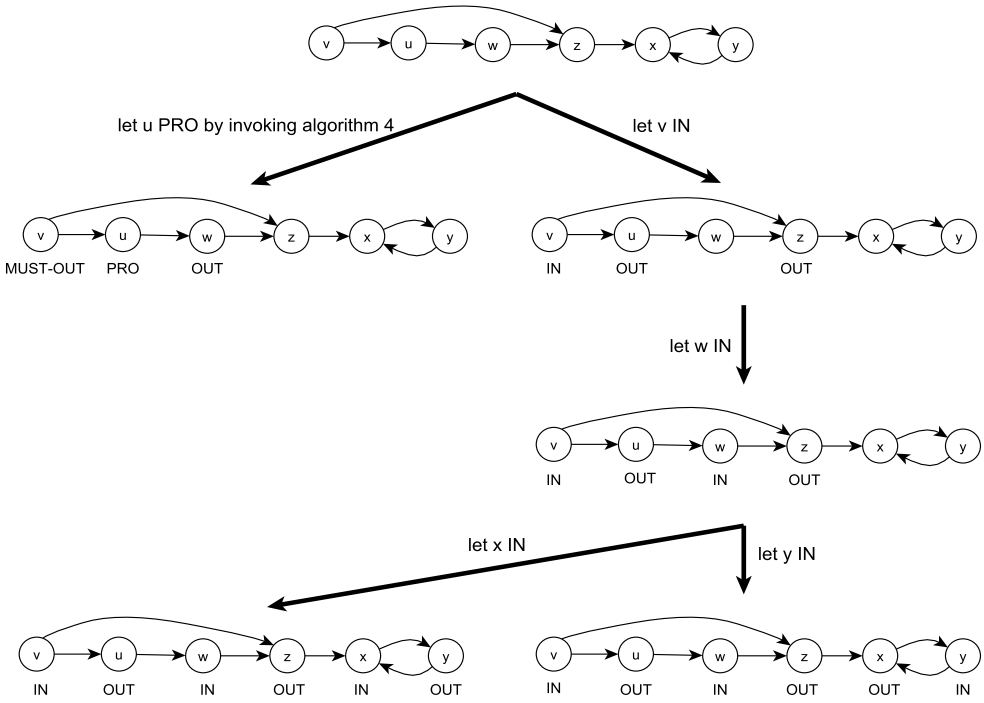

Figure 3.5: Deciding the skeptical acceptance of the argument w by using algorithm 13.

argument system depicted in figure 3.3. Note that figure 3.5 grows from left to right. The proof of algorithm 13 would be in two parts. The first part, which directly follows, is about showing that a given argument is not skeptically accepted if the argument is attacked by a credulously accepted argument, while the second part would be identical to the proof of algorithm 10.

3.5 The Advantage of these Algorithms over Existing Algorithms

3.5.1 The Algorithms of Cayrol et al.

We start by highlighting the main differences between algorithm 12 and the algorithm of Cayrol et al [28] (abbreviated by CAYCred) for the decision problem of credulous acceptance. Notice that CAYCred makes use of three labels: PRO, OPP and OUT. We use PRO/OPP in the same way CAYCred does. However, CAYCred labels an argument x OUT on three occasions: First, if x is attacked by a PRO argument; Second, if x attacks a PRO argument; Third, if x cannot be in an admissible set with the current PRO arguments. As we demonstrate, it is more efficient to use a different label on each distinct occasion. This is exactly what algorithm 12 does where we put in service OUT on the first occasion, MUST_OUT on the second occasion and IGNORED on the third

Algorithm 13: Deciding the skeptical proof of an argument x in an argument system (A, R).

1 $N(y) \equiv |\{z \in A \mid (y,z) \in R\}|$;
2 $Lab : A \rightarrow \{IN, OUT, MUST\_OUT, IGNORED, UNDEC\}$; $Lab \leftarrow \phi$;
3 **foreach** $y \in A$ **do** $Lab \leftarrow Lab \cup \{(y, UNDEC)\}$;
4 $E_{preferred} \subseteq 2^A$; $E_{preferred} \leftarrow \phi$;
5 **if** $\nexists (y,x) \in R$ **then**
6 x is skeptically proved by $\{x\}$; exit;
7 **foreach** $(y,x) \in R$ **do**
8 **invoke** algorithm 12 passing on A, R and y;
9 **if** *algorithm 12 decided that y is credulously accepted* **then**
10 x is not skeptically accepted; exit;
11 **call** decide-skeptical-acceptance(Lab);
12 **if** $E_{preferred} \neq \phi$ **then**
13 x is skeptically proved by $E_{preferred}$;

14 **procedure** decide-skeptical-acceptance(Lab)
15 **while** $\exists y \in A : Lab(y) = UNDEC$ **do**
16 select $y \in A$ s.t. $Lab(y) = UNDEC$ and $\forall (w,y) \in R$ $(Lab(w) \in \{OUT, MUST\_OUT\})$, otherwise select $y \in A$ s.t. $Lab(y) = UNDEC$ and $\forall z \in A : Lab(z) = UNDEC$, $N(y) \geq N(z)$;
17 $Lab' \leftarrow Lab$; $Lab'(y) \leftarrow IN$;
18 **foreach** $(y,z) \in R$ **do**
19 $Lab'(z) \leftarrow OUT$;
20 **foreach** $(z,y) \in R$ **do**
21 **if** $Lab'(z) \in \{IGNORED, UNDEC\}$ **then**
22 $Lab'(z) \leftarrow MUST\_OUT$;
23 **if** $\nexists (w,z) \in R : Lab'(w) = UNDEC$ **then**
24 $Lab(y) \leftarrow IGNORED$;
25 **goto** line 15;
26 **call** decide-skeptical-acceptance(Lab');
27 **if** $\exists (z,y) \in R : Lab(z) \in \{IGNORED, UNDEC\}$ **then**
28 $Lab(y) \leftarrow IGNORED$;
29 **else**
30 $Lab \leftarrow Lab'$;
31 **if** $\nexists y \in A : Lab(y) = MUST\_OUT$ **then**
32 $S \leftarrow \{y \in A \mid Lab(y) = IN\}$;
33 **if** $\nexists T \in E_{preferred} : S \subseteq T$ **then**
34 $E_{preferred} \leftarrow E_{preferred} \cup \{S\}$;
35 **if** $Lab(x) \neq IN$ **then**
36 $E_{preferred} \leftarrow \phi$;
37 x is not skeptically accepted; terminate and exit;
38 **end procedure**

occasion. To see the profit of our labeling scheme consider the following. CAYCred stops exploring further and backtracks if and only if there is $x \in A$ *s.t.* x attacks a PRO argument and for each $(z,x) \in R$ z is OUT. This stop condition is checked every time an argument is labeled PRO. Conversely, algorithm 12 backtracks if and only if a MUST_OUT argument is not attacked by an UNDEC argument. This backtrack condition is checked every time an argument is labeled MUST_OUT. Observe that searching for an argument attacking a PRO argument in CAYCred runs in the order of $|R|$ while looking for a MUST_OUT argument in algorithm 12 runs in the order of $|A|$; as we noted earlier typically $|R| > |A|$. Most importantly, CAYCred selects an argument to be PRO arbitrarily while algorithm 12 uses an effective heuristic rule that picks up an UNDEC argument attacked only by OPP, OUT or MUST_OUT arguments, otherwise the rule picks an UNDEC argument x such that $|\{x\}^+|$ is maximal, and so, the running time is improved as suggested by the experiments that we present shortly. Regarding the IGNORED label, the objective is to discriminate, and then to avoid, those arguments that previously failed to be in an admissible set with the current PRO arguments. The merit of the IGNORED label is also captured by CAYCred through the OUT label.

Concerning the decision problem of skeptical acceptance, the idea of the algorithm of Cayrol et al. [28] (CAYSkep for short) is based on an argument x not being skeptically accepted if at least one of two conditions holds:

1. x is attacked by a credulously accepted argument z (where z is decided by using CAYCred).

2. There exists an admissible set that does not contain x and cannot be expanded into one that contains it.

Otherwise, x is skeptically accepted provided that there exists an admissible set that contains x. Notice that, given the admissibility of the empty set and condition 2, it suffices to find just *one* admissible set containing x to ensure - conditions 1 & 2 having reported negatively - that x is skeptically accepted. Regarding condition (1), we commented earlier about the efficiency of CAYCred versus algorithm 12. In deciding condition (2), CAYSkep uses two labels IN and OUT, and so, an argument y is labeled IN to indicate that y might be in an admissible set. The usage of the OUT label is described earlier in the discussion on CAYCred. To check whether $S \subseteq A$ is an admissible set that can be expanded into one that contains the argument in question or not, CAYSkep verifies that S is maximally admissible in $\{y \in A \mid Lab(y) \in \{IN, OUT\}\}$. Such verification is relatively expensive, and thus, it is avoided by algorithm 13. Recall that algorithm 13 decides that an admissible set is maximal if and only if the set is not a subset of any previously decided preferred extension.

Table 3.11: Algorithm of Cayrol et al. for credulous acceptance versus algorithm 12, argument systems were generated by algorithm 11.

$\|A\|$	range of $\|R\|$	Algorithm of Cayrol et al.		Algorithm 12	
		α_{time}	$\alpha_{attacks}$	α_{time}	$\alpha_{attacks}$
55	55-1613	36.26	1,251,284.28	17.27	104,083.12
60	60-2028	52.60	2,031,521.23	19.80	167,708.98
65	97-2335	92.70	3,596,327.17	24.00	277,111.35
70	70-2654	120.70	5,399,152.38	30.70	413,788.19
75	75-2985	188.60	8,807,724.16	39.60	652,614.29
80	80-3687	277.20	13,099,196.22	63.00	1,085,469.36
85	85-3774	427.30	21,043,549.30	92.40	1,753,413.95
90	90-4374	617.80	31,069,987.87	129.50	2,582,447.94
95	95-4942	802.80	40,306,006.16	155.50	3,078,886.39
100	100-5410	1,326.80	67,985,810.32	245.80	5,251,456.56

Table 3.12: Algorithm of Cayrol et al. for credulous acceptance versus algorithm 12, argument systems were generated by setting attacks with a probability of $\frac{2\times log_e|A|}{|A|}$.

$\|A\|$	Algorithm of Cayrol et al.		Algorithm 12	
	α_{time}	$\alpha_{attacks}$	α_{time}	$\alpha_{attacks}$
55	70.30	3,203,294.39	18.20	167,594.91
60	121.20	5,717,097.24	30.10	309,147.83
65	211.20	9,923,949.12	51.10	542,240.70
70	365.60	17,746,378.47	86.40	948,694.78
75	621.30	30,285,694.59	148.60	1,645,812.26
80	1,013.20	50,020,197.35	260.90	2,924,630.80
85	1,852.20	91,931,844.61	479.90	5,328,965.38
90	3,055.00	152,084,077.32	771.80	8,493,395.25
95	5,009.10	255,574,227.54	1,291.40	14,701,867.73
100	8,146.10	416,991,952.56	2,316.70	26,152,189.79

We conducted experiments to show the efficiency of algorithms 12 and 13 in comparison with the algorithms of Cayrol et al. for credulous and skeptical acceptance. In subsection 3.3.1 we described the settings of our experiments. Concerning the experimental results, tables 3.11, 3.12, 3.13 and 3.14 suggest that algorithms 12 and 13 are more efficient than the algorithms of Cayrol et al.

3.5.2 The Algorithms of Thang et al.

The algorithm of Thang et al. [95] (abbreviated by ThCred) for the decision problem of credulous acceptance is based on classifying arguments into four sets: P, O, SP and SO. As an initial step, the argument in question is added to SP and P while O and SO are empty. Next, the following three operations are applied iteratively s.t. in every iteration one or more tuples of (P, O, SP, SO) might be generated.

Op1 If there is some $x \in P$ s.t. $SP \cap \{x\}^- = \emptyset$ then x is removed from P and every

Table 3.13: Algorithm of Cayrol et al. for skeptical acceptance versus algorithm 13, argument systems were generated by algorithm 11.

$\|A\|$	range of $\|R\|$	Algorithm of Cayrol et al.		Algorithm 13	
		α_{time}	$\alpha_{attacks}$	α_{time}	$\alpha_{attacks}$
16	16-159	86.50	1,182,283.26	28.20	2,554.26
17	17-200	231.70	3,933,513.50	22.40	3,186.59
18	18-200	679.50	13,680,403.49	20.10	5,092.02
19	19-241	626.20	12,444,896.57	27.00	4,994.59
20	20-237	3,088.20	61,452,174.66	25.60	7,328.86
21	21-280	2,829.80	45,166,810.64	22.70	7,068.00
22	22-264	7,645.50	158,528,969.88	24.40	8,382.35
23	23-327	14,247.70	301,829,977.25	25.00	11,295.71
24	24-357	51,162.70	1,114,893,329.39	23.00	17,736.67
25	25-365	97,529.40	2,014,486,555.51	24.20	17,308.47

Table 3.14: Algorithm of Cayrol et al. for skeptical acceptance versus algorithm 13, argument systems were generated by setting attacks with a probability of $\frac{2 \times log_e |A|}{|A|}$.

$\|A\|$	Algorithm of Cayrol et al.		Algorithm 13	
	α_{time}	$\alpha_{attacks}$	α_{time}	$\alpha_{attacks}$
16	88.90	1,646,992.36	5.90	2,731.00
17	251.00	4,812,274.54	5.60	4,125.40
18	1,123.80	24,160,850.86	5.70	6,154.13
19	1,196.60	25,971,371.74	7.10	6,108.26
20	2,803.40	61,296,848.52	9.30	8,077.34
21	5,400.80	121,497,830.73	7.30	9,751.24
22	14,353.40	321,656,752.02	10.30	13,304.90
23	22,277.60	496,105,603.76	9.50	13,151.45
24	95,019.60	2,214,991,030.88	14.70	20,753.46
25	161,399.10	2,282,838,566.26	27.90	21,620.36

$y \in \{x\}^- \setminus SO$ is added to O.

Op2 An argument x is added to SP and P if and only if there exists $\{x\}^+ \cap O \neq \emptyset$ and $x \notin O \cup SO$.

Op3 An argument y is moved from O to SO if $\{y\}^- \cap SP \neq \emptyset$.

Hence, ThCred at any time might have more than one tuple of (P, O, SP, SO). This reflects that ThCred explores the admissibility of different subsets of A. ThCred reports that the argument in question is credulously accepted if and only if there exists a tuple (P, O, SP, SO) s.t. P and O are both empty. Otherwise, the argument is not credulously accepted. To compare with algorithm 12, we stress three issues.

Firstly, ThCred algorithm might reconsider an argument x to be added to SP and P although x might already have failed to be in an admissible set with the same, current arguments in SP. Recall that algorithm 12 utilizes the IGNORED label to designate an argument x that failed to be in an admissible set, and so, x is avoided in future computations.

Secondly, ThCred might add arguments to O despite being attacked by arguments in SP. This eventually might waste time because ThCred might unnecessarily try further arguments to be added to SP and P to counter the newly added arguments to O. In algorithm 12, this situation is avoided by using the OUT label s.t. as soon as an argument x is labeled PRO, every argument that is attacked by x will be labeled OUT. Recall that algorithm 12 explores MUST_OUT arguments, whereas OUT arguments are disregarded because they are already attacked by a PRO argument.

Thirdly, ThCred does not feature heuristics or pruning machineries to accelerate the search process while algorithm 12 deploys pruning mechanisms and an effective heuristic rule that selects arguments for transitions as we showed in section 3.1.

Regarding skeptical acceptance, the algorithm of Thang et al. [95] (THSkep for short) relies for its correctness on the concept of a *complete base* (for x). A *base*, $\mathcal{B}$ for x being a set of admissible sets $\mathcal{B} = \{S_1, S_2, ..., S_n\}$ each of which contains x, and such that for every preferred extension, E *containing* x, there exists $S \in \mathcal{B}$ with $S \subseteq E$. A base $\mathcal{B}$ is *complete* if for *every* preferred extension, E, there is some $S \in \mathcal{B}$ for which $S \subseteq E$. The process of verifying skeptical acceptance of x is shown to be equivalent to identifying a complete base for x. Thus the skeptical proof of x consists of such a base and the efficiency of THSkep is determined not only by the performance of THCred, since THSkep depends on THCred in searching for admissible sets, but also on the efficiency with which a candidate collection can be validated as a complete base: this approach is not that adopted within algorithm 13.

We conducted experiments to show the efficiency of algorithms 12 and 13 in comparison with the algorithms of Thang et al. for credulous and skeptical acceptance. In subsection 3.3.1 we described the settings of our experiments. Concerning the experi-

Table 3.15: Algorithm of Thang et al. for credulous acceptance versus algorithm 12, argument systems were generated by using algorithm 11.

$\|A\|$	range of $\|R\|$	Algorithm of Thang et al.		Algorithm 12	
		α_{time}	$\alpha_{attacks}$	α_{time}	$\alpha_{attacks}$
26	26-407	105.90	156,842.93	12.50	3,788.83
27	27-406	182.50	263,501.54	13.60	4,312.80
28	28-480	334.70	463,823.87	11.30	5,193.49
29	29-545	534.40	759,273.63	11.20	5,714.35
30	30-540	726.80	1,025,980.91	11.00	6,807.88
31	34-587	970.30	1,304,047.51	13.20	7,803.19
32	44-585	1,003.60	1,405,827.47	11.40	8,704.97
33	33-722	1,661.10	2,115,693.68	11.20	9,478.59
34	34-677	3,523.30	4,429,569.59	11.80	11,775.78
35	35-763	4,689.19	5,414,065.42	12.80	12,468.76

Table 3.16: Algorithm of Thang et al. for credulous acceptance versus algorithm 12, argument systems were generated by setting attacks with a probability of $\frac{2\times log_e|A|}{|A|}$.

$\|A\|$	Algorithm of Thang et al.		Algorithm 12	
	α_{time}	$\alpha_{attacks}$	α_{time}	$\alpha_{attacks}$
26	886.40	1,188,915.47	3.50	3,986.67
27	1,274.60	1,587,213.93	4.30	4,349.90
28	1,848.10	2,230,412.10	4.40	5,006.85
29	3,217.40	3,570,786.55	4.60	5,724.25
30	6,447.80	6,761,709.98	7.10	6,757.64
31	7,840.10	8,302,807.05	5.90	7,834.19
32	13,063.00	12,916,358.74	7.20	8,586.04
33	20,009.50	19,562,812.87	2.30	10,005.02
34	48,028.00	45,190,073.97	8.30	12,741.22
35	109,077.60	80,256,368.60	13.40	14,289.45

mental results, tables 3.15, 3.16, 3.17 and 3.18 indicate that our algorithms outperform the algorithms of Thang et al.

3.5.3 The Algorithm of Verheij

Verheij [97] presented an algorithm for the credulous acceptance problem. Verheij classifies arguments into two sets J and D. Initially, J contains the argument in question while D is empty. Then, two functions are repeatedly executed on every pair of (J, D). The first function is

$$ExtendByAttack((J,D)) \equiv \{(J,D') \mid D' = D \cup J^{-}\}$$

The second function $ExtendByDefence((J,D))$ is given by,

$$\{(J',D) \mid J' \text{is a conflict free, minimal superset of } J,\ s.t.\ \forall y \in D\ \exists x \in J' \cap \{y\}^{-}\}$$

Table 3.17: Algorithm of Thang et al. for skeptical acceptance versus algorithm 13, argument systems were generated by using algorithm 11.

$\|A\|$	range of $\|R\|$	Algorithm of Thang et al.		Algorithm 13	
		α_{time}	$\alpha_{attacks}$	α_{time}	$\alpha_{attacks}$
11	11-81	20.20	3,744.24	12.10	567.72
12	12-94	29.20	4,463.72	11.80	937.50
13	13-108	58.80	8,971.51	12.40	1,322.93
14	14-125	95.90	11,680.88	11.50	1,479.11
15	15-129	177.80	17,659.00	11.00	2,154.33
16	16-182	320.30	22,933.12	12.40	2,871.67
17	17-173	844.80	46,866.46	11.20	2,629.22
18	18-195	1,538.80	62,819.86	11.90	4,309.18
19	19-234	3,597.80	96,327.30	11.60	5,181.75
20	20-232	6,539.90	124,679.03	13.20	6,256.34

Table 3.18: Algorithm of Thang et al. for skeptical acceptance versus algorithm 13, argument systems were generated by setting attacks with a probability of $\frac{2 \times log_e|A|}{|A|}$.

$\|A\|$	Algorithm of Thang et al.		Algorithm 13	
	α_{time}	$\alpha_{attacks}$	α_{time}	$\alpha_{attacks}$
11	26.90	8,204.52	2.10	846.01
12	36.50	10,781.42	2.40	1,214.93
13	84.80	25,630.05	1.80	1,502.19
14	169.90	39,257.39	2.30	1,718.46
15	302.90	54,498.72	2.40	2,353.65
16	754.80	93,401.01	6.40	3,274.23
17	2,384.00	156,534.51	6.20	3,857.90
18	5,559.50	268,672.44	8.10	4,568.57
19	8,961.40	332,887.49	6.40	6,044.24
20	18,586.40	502,895.22	8.00	8,394.86

Table 3.19: Algorithm of Verheij for credulous acceptance versus algorithm 12, argument systems were generated by using algorithm 11.

$\|A\|$	range of $\|R\|$	Algorithm of Verheij		Algorithm 12	
		α_{time}	$\alpha_{attacks}$	α_{time}	$\alpha_{attacks}$
21	21-245	136.30	409,670.86	19.20	1,932.56
22	22-297	117.10	513,117.24	19.60	2,086.22
23	23-284	227.50	980,685.42	19.80	2,647.67
24	34-356	318.30	1,537,036.22	20.20	2,983.99
25	25-356	553.60	2,615,823.22	17.70	3,181.49
26	26-387	790.10	3,654,956.54	18.70	3,937.03
27	27-411	1,386.40	5,925,777.07	18.90	4,390.59
28	28-458	2,778.30	11,754,982.28	19.70	5,437.08
29	29-501	4,049.50	15,948,103.70	19.90	5,851.49
30	30-576	5,693.94	25,065,980.18	19.80	7,127.35

Next, if there exists (J', D') and (J, D) such that $J' = J$ and $D' = D$ then the argument in question is credulously proved by (J', D'). At any stage if no new pair (J', D') is produced from applying the two functions on all current pairs of (J, D) then the argument is not accepted. To evaluate the performance of Verheij's approach in contrast to algorithm 12 we consider four efficiency matters.

Firstly, notice the price of finding a minimal defense set J' against the arguments in D, see the definition of $ExtendByDefence$ earlier. This is totally bypassed by algorithm 12.

Secondly, Verheij might extend D by adding superfluously arguments already attacked by arguments in J. This might worsen the efficiency of computing J' where more arguments in D might lead to more possible defense sets, and consequently, finding a minimal defense set J' would be more difficult. In algorithm 12 this situation is handled by using the OUT label designating arguments that are attacked by PRO arguments, and thus no further action is taken regarding the OUT arguments.

Thirdly, Verheij might extend J by adding arguments that already failed to form an admissible set with the same, current arguments in J. Perceive that algorithm 12 takes advantage of the IGNORED label to characterize the arguments that cannot make up an admissible set with the PRO arguments, and in consequence, IGNORED arguments will not be re-examined later.

Fourthly, the algorithm of Verheij does not employ heuristics or pruning techniques to enhance the search progression whereas algorithm 12 makes use of such performance boosters as we explained earlier in section 3.1.

We conducted experiments to show the efficiency of algorithms 12 in comparison with the algorithm of Verheij for credulous acceptance. In subsection 3.3.1 we described the settings of our experiments. Concerning the experimental results, tables 3.19 and 3.20 show that algorithm 12 is faster than the algorithm of Verheij.

Table 3.20: Algorithm of Verheij for credulous acceptance versus algorithm 12, argument systems were generated by setting attacks with a probability of $\frac{2\times log_e|A|}{|A|}$.

$\|A\|$	Algorithm of Verheij		Algorithm 12	
	α_{time}	$\alpha_{attacks}$	α_{time}	$\alpha_{attacks}$
21	249.50	1,215,070.67	2.10	1,777.89
22	454.70	2,205,743.88	3.30	2,085.41
23	781.00	3,672,245.30	4.00	2,457.69
24	1,369.90	6,190,955.17	3.30	2,949.88
25	2,166.00	9,866,104.89	3.10	3,312.02
26	3,834.60	16,921,018.16	5.30	3,796.28
27	6,970.90	29,189,155.84	4.10	4,548.34
28	10,679.60	42,854,169.54	3.70	4,829.02
29	18,789.90	73,944,520.95	5.40	5,663.20
30	33,697.20	124,127,437.05	7.40	6,614.57

3.6 Summary

In this chapter we developed a new algorithm for enumerating preferred extensions of an argument system. We have shown that the new algorithm computes extensions faster than the existing algorithms of [35, 78]. In the next chapter we design algorithms for enumerating extensions under a number of argumentation semantics other than the preferred semantics. Indeed, argumentation semantics can be defined by using labellings as well as extensions, see e.g. [6, 64]. In this work we present new algorithms that make use of labels as an algorithmic vehicle rather than introducing new label-based semantics. In fact, Doutre and Mengin have developed their labeling-based algorithm without elaborating label-based semantics [35]. On the other hand, in [78] Modgil and Caminada introduced label-based semantics as a foundation for their algorithm. The notion of labeling is also employed in dialog games see e.g. [21].

Likewise, we presented algorithms that decide the credulous and skeptical acceptance problems without explicitly enumerating all preferred extensions. An added feature of the developed algorithms is the production of proofs as to why an argument is accepted. We have shown, analytically and empirically, that our algorithms are more efficient than the existing algorithms of [28, 95, 97]. Some authors call the algorithms that yield proofs "dialectical proof procedures" referring to the fact that a proof of an accepted argument might be, roughly speaking, defined by the arguments put forward during a dialog between two parties. In fact, argumentation semantics can be defined by using the dialog notion (see e.g. [63, 100, 43, 75]). Hence, Cayrol et al. [28] set dialogs under preferred semantics as a means for presenting their algorithms. However, Thang et al. [95] make use of so-called "dispute trees" to pave the way for introducing their algorithms, while Verheij [97] defined his algorithm by employing the notion of "labellings" rather than specifying formal dialogs. Furthermore, argument-based

dialogs have been extensively studied as a backbone for interactions between agents in multi-agent systems, see e.g. [73] for an overview.

Broadly, there are several works around computing decision problems in argument systems. [99] discussed algorithmically the efficiency of deciding minimally admissible sets. [80] modified the algorithm of Doutre and Mengin [35] to compute preferred extensions for the extended systems of [81] so-called "argument systems with sets of attacking arguments". The work of [36] specifies dialogs for skeptical proofs under preferred semantics. The algorithms of [25, 27] find semi stable, respectively stage, extensions. Another line of research concerns encoding decision problems of argument systems into other formalisms and then solving them by using a respective solver, see for example [16, 82, 56, 2, 51, 58]. The work of [70] examines approximation versus exact computations in the context of argument systems, whereas the experiments of [12] evaluate the effect of splitting an argument system on the computation of the preferred extensions. The work of [71] shows how to partially reevaluate the acceptance of arguments if R changes. From a computational theoretical perspective, the decision problems of skeptical and credulous acceptance under preferred semantics are believed to be intractable, see e.g. [34, 40, 87].

Chapter 4

New Algorithms For a Selection of Argumentation Semantics

In this chapter we develop algorithms for enumerating extensions under a number of prevalent argumentation semantics, which we defined in section 1.4. In particular, we design algorithms for enumerating stable extensions in section 4.1, complete extensions in section 4.2, stage extensions in section 4.3, and semi stable extensions in section 4.4. In section 4.5 we present an algorithm for deciding the ideal extension while in section 4.6 we offer an implementation to an existing algorithm for deciding the grounded extension. We explore the efficiency of these algorithms by comparing with ASPARTIX [56], which is an Answer Set Programming encoding for several argumentation semantics. We used the DLV system [69] (release 21/12/2011) in solving the encodings. Lastly, we close the chapter in section 4.7.

4.1 A New Algorithm for Stable Semantics

Algorithm 14 decides all stable extensions. Indeed, algorithm 14 is a modification of algorithm 10, which decides preferred extensions. In algorithm 10 we find a preferred extension $pref$ if and only if for every $x \in A$, x is not UNDEC nor MUST_OUT and $pref$ is not a subset of a previously decided preferred extension. However, in algorithm 14 we encounter a stable extension if and only if for each $x \in A$, x is not UNDEC nor MUST_OUT nor IGNORED. Furthermore, algorithm 14 applies an additional pruning strategy (see lines 21 and 29) such that we skip ignoring an argument y if and only if there does not exist $(z, y) \in R$ $s.t.$ z is UNDEC. To see why, note that we get a stable extension if and only if for each $x \in A$, x is not UNDEC nor MUST_OUT nor IGNORED. Thus, if we ignore an argument y that will stay IGNORED, since there is no $(z, y) \in R$ $s.t.$ z is UNDEC, then it is more efficient to avoid ignoring such an argument in the first place. Tables 4.1 and 4.2 show the performance of algorithm 14 versus DLV system solving the ASPARTIX encoding. Recall that in this thesis we report the average elapsed time in milliseconds. Note that in table 4.2 ASPARTIX performs better than algorithm 14

when the probability is $\frac{2\times log_e|A|}{|A|}$. However, in light of the results of table 4.1 algorithm 14 on average outperforms ASPARTIX if we consider sparse argument systems and dense ones altogether. Table 4.2 reflects that algorithm 14 performs better when the input argument system is more dense; this performance transition seems to occur around the probability of $\frac{3\times log_e|A|}{|A|}$. Notice that as $|R|$ is getting larger, labeling an argument x with IN will expel a larger number of arguments (i.e. $\{x\}^+ \cup \{x\}^-$) thereby leaving a fewer UNDEC arguments for the possibility to be included in the candidate extension, which means a fewer steps until a decision can be made about whether the arguments labeled IN make up an extension or not.

4.2 A New Algorithm for Complete Semantics

Algorithm 15 decides all complete extensions. Indeed, algorithm 15 is a modification of algorithm 10, which decides preferred extensions. In algorithm 10 we find a preferred extension $pref$ if and only if for each $x \in A$, x is not UNDEC nor MUST_OUT and $pref$ is not a subset of a previously decided preferred extension. However, in algorithm 15 we encounter a complete extension if and only if

$$\forall x \in A,\ x \textit{ is not MUST\_OUT and}$$
$$\nexists\, z \in A :\ z \textit{ is IGNORED or UNDEC with } \forall (y,z) \in R\ y \textit{ is OUT}.$$

Tables 4.3 and 4.4 show the performance of algorithm 15 versus ASPARTIX.

4.3 A New Algorithm for Stage Semantics

Algorithm 16 decides all stage extensions. In particular, algorithm 16 decides candidate conflict free subsets of A (see lines 14-29) in the same way (i.e. using the same set of labels) algorithm 10 does in deciding preferred extensions. Recall that algorithm 10 actually enumerates admissible sets rather than conflict free sets. Thus, algorithm 10 decides that IN arguments make up an admissible set if and only if for each $x \in A$ x is not UNDEC nor MUST_OUT while algorithm 16 reports IN arguments as a conflict free set if and only if for each $x \in A$, x is not UNDEC. Then, for every reported conflict free set S algorithm 16 also determines $S' \equiv \{x \in A \mid x \text{ is OUT}\}$. After accumulating all candidate $S \cup S'$, algorithm 16 decides that a conflict free set S is a stage extension if and only if $S \cup S'$ is maximal, see lines 7-11.

Heuristics and pruning strategies used in semantics that are based on admissible sets will not be applicable to stage semantics, which are based on conflict free sets. Therefore, as a pruning strategy we skip (see line 27 of algorithm 16) ignoring an argument y if and only if for each $z \in \{y\}^+ \cup \{y\}^-$, z is OUT or MUST_OUT or IGNORED. Note that labeling such y IGNORED is unnecessary as we explain shortly. On selecting the next UNDEC argument to be labeled IN, there are two options. For the first option, denoted by Heu_1, we take into account the following rule:

Algorithm 14: Enumerating all stable extensions of an argument system (A, R).

1 $N(y) \equiv |\{z \in A \mid (y,z) \in R\}|$;
2 $Lab : A \rightarrow \{IN, OUT, MUST\_OUT, IGNORED, UNDEC\}$; $Lab \leftarrow \phi$;
3 **foreach** $x \in A$ **do** $Lab \leftarrow Lab \cup \{(x, UNDEC)\}$;
4 $E_{stable} \subseteq 2^A$; $E_{stable} \leftarrow \phi$;
5 **call** find-stable-extensions(Lab);
6 **report** E_{stable} is the set of all stable extensions;

7 **procedure** find-stable-extensions(Lab)
8 **while** $\exists y \in A : Lab(y) = UNDEC$ **do**
9 select $y \in A$ s.t. $Lab(y) = UNDEC$ and $\forall (z,y) \in R$ $(Lab(z) \in \{OUT, MUST\_OUT\})$, otherwise select $y \in A$ s.t. $Lab(y) = UNDEC$ and $\forall z \in A : Lab(z) = UNDEC$, $N(y) \geq N(z)$;
10 $Lab' \leftarrow Lab$;
11 $Lab'(y) \leftarrow IN$;
12 **foreach** $(y,z) \in R$ **do** $Lab'(z) \leftarrow OUT$;
13 **foreach** $(z,y) \in R$ **do**
14 **if** $Lab'(z) \in \{IGNORED, UNDEC\}$ **then**
15 $Lab'(z) \leftarrow MUST\_OUT$;
16 **if** $\nexists (w,z) \in R : Lab'(w) = UNDEC$ **then**
17 **if** $\exists (v,y) \in R : Lab(v) = UNDEC$ **then**
18 $Lab(y) \leftarrow IGNORED$;
19 **goto** line 8;
20 **else**
21 return;
22 **call** find-stable-extensions(Lab');
23 **if** $\exists (z,y) \in R : Lab(z) = UNDEC$ **then**
24 $Lab(y) \leftarrow IGNORED$;
25 **else**
26 **if** $\nexists (z,y) \in R : Lab(z) = IGNORED$ **then**
27 $Lab \leftarrow Lab'$;
28 **else**
29 return;
30 **if** $\nexists y \in A : Lab(y) \in \{MUST\_OUT, IGNORED\}$ **then**
31 $S \leftarrow \{y \in A \mid Lab(y) = IN\}$;
32 $E_{stable} \leftarrow E_{stable} \cup \{S\}$;
33 **end procedure**

Algorithm 15: Enumerating all complete extensions of an argument system (A, R).

1 $N(y) \equiv |\{z \in A \mid (y,z) \in R\}|$;
2 $Lab : A \to \{IN, OUT, MUST\_OUT, IGNORED, UNDEC\}$; $Lab \leftarrow \phi$;
3 **foreach** $x \in A$ **do** $Lab \leftarrow Lab \cup \{(x, UNDEC)\}$;
4 $E_{complete} \subseteq 2^A$; $E_{complete} \leftarrow \phi$;
5 **call** find-complete-extensions(Lab);
6 **report** $E_{complete}$ is the set of all complete extensions;

7 **procedure** find-complete-extensions(Lab)
8 **if** $\nexists y \in A : Lab(y) = MUST\_OUT$ **then**
9 **if** $\nexists x \in A : Lab(x) \in \{IGNORED, UNDEC\}$ *and* $\forall (z,x) \in R\ Lab(z) = OUT$ **then**
10 $S \leftarrow \{w \in A \mid Lab(w) = IN\}$;
11 $E_{complete} \leftarrow E_{complete} \cup \{S\}$;
12 **while** $\exists y \in A : Lab(y) = UNDEC$ **do**
13 select $y \in A$ s.t. $Lab(y) = UNDEC \wedge \forall (z,y) \in R\ (z \in \{OUT, MUST\_OUT\})$, otherwise select $y \in A$ s.t. $Lab(y) = UNDEC$ and $\forall z \in A : Lab(z) = UNDEC,\ N(y) \geq N(z)$;
14 $Lab' \leftarrow Lab$;
15 $Lab'(y) \leftarrow IN$;
16 **foreach** $(y,z) \in R$ **do** $Lab'(z) \leftarrow OUT$;
17 **foreach** $(z,y) \in R$ **do**
18 **if** $Lab'(z) \in \{IGNORED, UNDEC\}$ **then**
19 $Lab'(z) \leftarrow MUST\_OUT$;
20 **if** $\nexists (w,z) \in R : Lab'(w) = UNDEC$ **then**
21 $Lab(y) \leftarrow IGNORED$;
22 **goto** line 12;
23 **call** find-complete-extensions(Lab');
24 **if** $\exists (z,y) \in R : Lab(z) \in \{UNDEC, IGNORED\}$ **then**
25 $Lab(y) \leftarrow IGNORED$;
26 **else**
27 $Lab \leftarrow Lab'$;
28 **end procedure**

Table 4.1: The average elapsed time of algorithm 14 versus ASPARTIX, argument systems were generated by using algorithm 11.

$\|A\|$	Range of $\|R\|$	ASPARTIX	algorithm 14
61	61-1952	30.80	13.30
62	62-1842	31.30	12.90
63	94-2101	34.40	14.80
64	64-2208	34.10	14.30
65	65-2458	38.20	15.40
66	66-2334	39.30	15.00
67	67-2477	37.30	18.70
68	68-2526	41.10	17.50
69	100-2628	45.60	16.40
70	70-2795	50.80	19.80

Table 4.2: The average elapsed time of algorithm 14 versus ASPARTIX, argument systems were generated by setting attacks with a specific probability.

$\|A\|$	probability=$\frac{2\times log_e\|A\|}{\|A\|}$		probability=$\frac{3\times log_e\|A\|}{\|A\|}$		probability=$\frac{4\times log_e\|A\|}{\|A\|}$	
	ASPARTIX	Alg. 14	ASPARTIX	Alg. 14	ASPARTIX	Alg. 14
61	20.30	36.20	86.9	43.2	66.90	12.50
62	20.50	42.00	89.8	45.2	69.50	13.10
63	20.40	47.40	90.4	51.6	74.70	13.60
64	22.00	52.10	95.4	55.6	76.00	16.30
65	24.50	63.60	101.8	63.1	82.50	19.40
66	23.30	68.20	101.9	68.6	87.60	21.40
67	25.00	81.20	110	75.6	91.20	23.10
68	24.90	92.00	121.6	82.8	92.90	25.60
69	28.10	112.20	126.9	93.8	102.70	30.30
70	28.20	122.20	132.3	103.9	111.40	34.30

1. select an UNDEC argument y s.t. for each $z \in \{y\}^+ \cup \{y\}^-$, z is OUT or MUST_OUT or IGNORED.

2. otherwise select an UNDEC argument y such that $|\{y\}^+|$ is maximal.

For the second possibility, denoted by Heu_2, we consider the following rule:

1. select an UNDEC argument y s.t. for each $z \in \{y\}^+ \cup \{y\}^-$, z is OUT or MUST_OUT or IGNORED.

2. otherwise select an UNDEC argument y such that $|\{y\}^+| + |\{y\}^-|$ is maximal.

The aim of the first part of Heu_1 & Heu_2, which is identical in both selection rules, is to maximize the gain of the pruning strategy that skips ignoring y, i.e. the selected

Table 4.3: The average elapsed time of algorithm 15 versus ASPARTIX, argument systems were generated by using algorithm 11.

$\|A\|$	Range of $\|R\|$	ASPARTIX	Algorithm 15
56	56-1849	43.90	15.10
57	57-1786	47.90	16.50
58	91-1880	52.70	20.80
59	59-1893	57.80	12.90
60	60-2047	61.00	20.30
61	61-1989	61.10	19.10
62	62-2037	68.60	24.10
63	63-2275	75.10	21.30
64	92-2160	77.60	33.20
65	65-2154	86.00	37.70

Table 4.4: The average elapsed time of algorithm 15 versus ASPARTIX, argument systems were generated by setting attacks with a specific probability.

$\|A\|$	probability=$\frac{2\times log_e\|A\|}{\|A\|}$		probability=$\frac{4\times log_e\|A\|}{\|A\|}$	
	ASPARTIX	Algorithm 15	ASPARTIX	Algorithm 15
56	30.20	34.70	137.70	9.90
57	33.10	41.60	167.10	10.50
58	38.60	47.60	173.70	10.20
59	40.30	57.00	189.20	10.10
60	44.60	72.80	198.90	10.70
61	42.90	76.50	221.00	14.20
62	44.70	90.80	245.20	17.70
63	53.50	107.90	283.00	19.50
64	54.90	127.10	303.70	22.70
65	62.70	150.30	337.40	25.30

argument in the the first part of the rules, based on the following property: if a conflict free set S will be captured while such y is ignored then $S \cup \{y\}$ is conflict free as well, and so, there is no need to ignore y in the first place. Consequently, the earlier we label such y IN, the bigger part of the search tree that will be bypassed. Turning to the second part of Heu_1 & Heu_2. Recall that the aim of heuristics in our algorithms is to accelerate achieving a goal state. In algorithm 16 a goal state is a conflict free set with a maximal range such that there is no $x \in A : x$ is UNDEC. So, we note Heu_2 is potentially more powerful, because, by maximizing the number of OUT/MUST_OUT arguments the number of UNDEC arguments is minimized more than Heu_1 which maximizes only the number of OUT arguments. See tabels 4.5 and 4.6 that reflect the efficiency of algorithm 16 by using Heu_1 and Heu_2 versus DLV solving ASPARTIX encodings.

Algorithm 16: Enumerating stage extensions of an argument system (A, R).

1 $N(y) \equiv |\{z \in A \mid (y,z) \in R\}|$;
2 $O(y) \equiv |\{z \in A \mid (z,y) \in R\}|$;
3 $Lab : A \rightarrow \{IN, OUT, MUST\_OUT, IGNORED, UNDEC\}$; $Lab \leftarrow \phi$;
4 **foreach** $x \in A$ **do** $Lab \leftarrow Lab \cup \{(x, UNDEC)\}$;
5 $E_{stage} : (A \rightarrow \{IN, OUT, MUST\_OUT, IGNORED, UNDEC\}) \times \mathbb{Z}$; $E_{stage} \leftarrow \phi$;
6 **call** find-conflict-free-sets(Lab);
`/* The next loop is to collect conflict free sets, those which have a maximal range */`
7 **foreach** $(Lab_1, i) \in E_{stage}$ **do**
8 **foreach** $(Lab_2, j) \in E_{stage} : j \neq i$ **do**
9 **if** $(|\{x : Lab_1(x) \in \{IN, OUT\}\}| \neq |\{z : Lab_2(z) \in \{IN, OUT\}\}| \vee Lab_1 = Lab_2)$ *and* $\forall y \in A : Lab_1(y) \in \{IN, OUT\}$ $(Lab_2(y) \in \{IN, OUT\})$ **then**
10 $E_{stage} \leftarrow E_{stage} \setminus \{(Lab_1, i)\}$;
11 continue to next iteration from line 7;
12 **foreach** $(Lab_1, i) \in E_{stage}$ **do**
13 **report** $\{x : Lab_1(x) = IN\}$ as a stage extension;

14 **procedure** find-conflict-free-sets(Lab)
15 **while** $\exists y \in A : Lab(y) = UNDEC$ **do**
16 select $y \in A$ s.t. $Lab(y) = UNDEC$ and $\forall z \in \{y\}^+ \cup \{y\}^-$ $(Lab(z) \in \{OUT, MUST\_OUT, IGNORED\})$, otherwise select $y : Lab(y) = UNDEC$ *satisfying* $\forall z : Lab(z) = UNDEC$, $(N(y) + O(y)) \geq (N(z) + O(z))$;
17 $Lab' \leftarrow Lab$;
18 $Lab'(y) \leftarrow IN$;
19 **foreach** $(y,z) \in R$ **do** $Lab'(z) \leftarrow OUT$;
20 **foreach** $(z,y) \in R$ **do**
21 **if** $Lab'(z) \in \{IGNORED, UNDEC\}$ **then**
22 $Lab'(z) \leftarrow MUST\_OUT$;
23 **call** find-conflict-free-sets(Lab');
24 **if** $\exists z \in \{y\}^+ \cup \{y\}^-$ *and* $Lab(z) = UNDEC$ **then**
25 $Lab(y) \leftarrow IGNORED$;
26 **else**
27 $Lab \leftarrow Lab'$;
28 $E_{stage} \leftarrow E_{stage} \cup \{(Lab, |E_{stage}| + 1)\}$;
29 **end procedure**

Table 4.5: The average elapsed time of algorithm 16 versus ASPARTIX, argument systems were generated by using algorithm 11.

$\|A\|$	Range of $\|R\|$	ASPARTIX	Algorithm 16 using Heu_1	Algorithm 16 using Heu_2
21	21-245	302.90	1.80	1.10
22	22-293	392.10	3.60	1.50
23	23-332	541.40	6.10	2.40
24	24-322	724.50	8.50	4.60
25	25-379	813.40	19.40	6.70
26	26-471	1,328.90	26.90	11.70
27	27-396	1,619.00	39.30	13.10
28	28-463	2,143.80	80.00	17.10
29	29-552	4,027.90	93.30	47.20
30	30-483	3,717.70	140.00	45.40

4.4 A New Algorithm for Semi Stable Semantics

Algorithm 17 decides all semi stable extensions. In particular, algorithm 17 decides candidate admissible sets (see lines 13-32) in the same way algorithm 10 does in deciding preferred extensions. However, for every decided admissible set S algorithm 17 also collects $S' \equiv \{x \in A \mid x \text{ is OUT}\}$. After accumulating all $S \cup S'$, algorithm 17 decides that an admissible set S is a semi stable extension if and only if $S \cup S'$ is maximal, see lines 6-10. Tables 4.7 and 4.8 show the performance of algorithm 17 versus ASPARTIX.

4.5 A New Algorithm for Ideal Semantics

Algorithm 18 decides the ideal extension. In particular, algorithm 18 decides candidate admissible sets (see lines 11-31) in the same way algorithm 10 does in deciding preferred extensions. However, in enumerating admissible sets algorithm 18 collects (line 29) $S \equiv \{x \mid \textit{ there is an admissible set } T \textit{ s.t. } x \in \{T\}^+\}$. After enumerating a set of admissible sets and having S determined, algorithm 18 decides that an admissible set T is the ideal extension if and only if for each $z \in T$ $z \notin S$, see lines 7-10. Note that we collect candidate admissible sets in a descending order, that is from the largest set to the smallest one. Table 4.9 shows the performance of algorithm 18. We did not compare with ASPARTIX since it is very slow such that we were not able to run ASPARTIX on quite large argument systems.

4.6 Deciding the Grounded Extension

Algorithm 19 can be seen as another implementation of the algorithm described in [78] for deciding the grounded extension. We report in tables 4.10 and 4.11 the efficiency

Algorithm 17: Enumerating semi stable extensions of an argument system (A, R).

1 $N(y) \equiv |\{z \in A \mid (y,z) \in R\}|$;
2 $Lab : A \rightarrow \{IN, OUT, MUST\_OUT, IGNORED, UNDEC\}$; $Lab \leftarrow \phi$;
3 **foreach** $x \in A$ **do** $Lab \leftarrow Lab \cup \{(x, UNDEC)\}$;
4 $E_{semistable} : (A \rightarrow \{IN, OUT, MUST\_OUT, IGNORED, UNDEC\}) \times \mathbb{Z}$; $E_{semistable} \leftarrow \phi$;
5 **call** find-admissible-sets(Lab);
`/* The next loop is to pick up admissible sets, those which have a maximal range */`
6 **foreach** $(Lab_1, i) \in E_{semistable}$ **do**
7 **foreach** $(Lab_2, j) \in E_{semistable} : j \neq i$ **do**
8 **if** $(|\{x : Lab_1(x) \in \{IN, OUT\}\}| \neq |\{z : Lab_2(z) \in \{IN, OUT\}\}| \vee Lab_1 = Lab_2)$ *and* $\forall y \in A : Lab_1(y) \in \{IN, OUT\}$ $(Lab_2(y) \in \{IN, OUT\})$ **then**
9 $E_{semistable} \leftarrow E_{semistable} \setminus \{(Lab_1, i)\}$;
10 continue to next iteration from line 6;
11 **foreach** $(Lab_1, i) \in E_{semistable}$ **do**
12 **report** $\{x : Lab_1(x) = IN\}$ as a semi stable extension ;

13 **procedure** find-admissible-sets(Lab)
14 **while** $\exists y \in A : Lab(y) = UNDEC$ **do**
15 select $y \in A$ with $Lab(y) = UNDEC$ and $\forall z \in \{y\}^- (z \in \{OUT, MUST\_OUT\})$, otherwise
select y *with* $Lab(y) = UNDEC$ *and* $\forall z : Lab(z) = UNDEC$, $N(y) \geq N(z)$;
16 $Lab' \leftarrow Lab$;
17 $Lab'(y) \leftarrow IN$;
18 **foreach** $(y,z) \in R$ **do** $Lab'(z) \leftarrow OUT$;
19 **foreach** $(z,y) \in R$ **do**
20 **if** $Lab'(z) \in \{IGNORED, UNDEC\}$ **then**
21 $Lab'(z) \leftarrow MUST\_OUT$;
22 **if** $\nexists (w,z) \in R : Lab'(w) = UNDEC$ **then**
23 $Lab(y) \leftarrow IGNORED$;
24 **goto** line 14;
25 **call** find-admissible-sets(Lab');
26 **if** $\exists (z,y) \in R : Lab(z) \in \{UNDEC, IGNORED\}$ **then**
27 $Lab(y) \leftarrow IGNORED$;
28 **else**
29 $Lab \leftarrow Lab'$;
30 **if** $\nexists y \in A : Lab(y) = MUST\_OUT$ **then**
31 $E_{semistable} \leftarrow E_{semistable} \cup \{(Lab, |E_{semistable}| + 1)\}$;
32 **end procedure**

Algorithm 18: Deciding the ideal extension of an argument system (A, R).

1 $N(y) \equiv |\{z \in A \mid (y,z) \in R\}|$;
2 $Lab : A \rightarrow \{IN, OUT, MUST\_OUT, IGNORED, UNDEC\}$; $Lab \leftarrow \phi$;
3 **foreach** $x \in A$ **do** $Lab \leftarrow Lab \cup \{(x, UNDEC)\}$;
4 $E_{ideal} : \mathbb{Z} \rightarrow 2^A$; $E_{ideal} \leftarrow \phi$;
5 $S \leftarrow \phi$;
`/* S will hold the arguments that are not in the ideal extension */`
6 **call** find-admissible-sets(Lab);
7 **foreach** $i :\ 1$ *to* $|E_{idea}|$ **do**
8 **if** $\forall x \in E_{ideal}(i)\ (x \notin S)$ **then**
9 **report** $E_{ideal}(i)$ is the ideal extension;
10 **exit**;

11 **procedure** find-admissible-sets(Lab)
12 **while** $\exists y \in A : Lab(y) = UNDEC$ **do**
13 select y *with* $Lab(y) = UNDEC$ *and* $\forall (z,y) \in R\ (z \in \{OUT, MUST\_OUT\})$, otherwise
select y *with* $Lab(y) = UNDEC$ *and* $\forall z : Lab(z) = UNDEC,\ N(y) \geq N(z)$;
14 $Lab' \leftarrow Lab$;
15 $Lab'(y) \leftarrow IN$;
16 **foreach** $(y,z) \in R$ **do** $Lab'(z) \leftarrow OUT$;
17 **foreach** $(z,y) \in R$ **do**
18 **if** $Lab'(z) \in \{IGNORED, UNDEC\}$ **then**
19 $Lab'(z) \leftarrow MUST\_OUT$;
20 **if** $\nexists (w,z) \in R : Lab'(w) = UNDEC$ **then**
21 $Lab(y) \leftarrow IGNORED$;
22 **goto** line 12;
23 **call** find-admissible-sets(Lab');
24 **if** $\exists (z,y) \in R : Lab(z) \in \{UNDEC, IGNORED\}$ **then**
25 $Lab(y) \leftarrow IGNORED$;
26 **else**
27 $Lab \leftarrow Lab'$;
28 **if** $\nexists y \in A : Lab(y) = MUST\_OUT$ **then**
29 $S \leftarrow S \cup \{x \in A \mid Lab(x) = OUT\}$;
30 $E_{ideal} \leftarrow E_{ideal} \cup \{(|E_{idea}| + 1, \{z \in A \mid Lab(z) = IN\})\}$;
31 **end procedure**

Table 4.6: The average elapsed time of algorithm 16 versus ASPARTIX, argument systems were generated by setting attacks with a specific probability.

$\|A\|$	probability=$\frac{2\times log_e\|A\|}{\|A\|}$		probability=$\frac{4\times log_e\|A\|}{\|A\|}$	
	ASPARTIX	Algorithm 16	ASPARTIX	Algorithm 16
21	220.80	0.20	112.30	0.30
22	303.30	0.20	142.50	0.70
23	389.90	0.00	184.40	0.20
24	540.20	0.00	218.90	2.90
25	784.40	0.50	286.30	0.10
26	1,069.80	0.40	330.50	0.80
27	1,494.10	3.40	462.50	0.30
28	2,153.20	7.60	608.30	0.20
29	3,057.20	10.20	882.50	0.40
30	4,468.90	12.30	1,288.50	0.20

Table 4.7: The average elapsed time of algorithm 17 versus ASPARTIX, argument systems were generated by using algorithm 11.

$\|A\|$	Range of $\|R\|$	ASPARTIX	Algorithm 17
36	36-780	224.80	1.20
37	37-766	226.60	1.60
38	38-794	284.30	2.10
39	39-894	293.80	1.60
40	40-818	338.50	3.30
41	41-916	375.40	3.40
42	63-907	428.70	5.60
43	43-1000	411.60	5.20
44	44-1160	453.70	5.70
45	45-1159	521.90	6.40

of algorithm 19 versus ASPARTIX.

4.7 Summary

We developed algorithms for enumerating extensions under stable semantics, complete semantics, stage semantics, semi stable semantics and ideal semantics. We have shown that these algorithms intersect with our algorithm for enumerating preferred extensions. Actually, all these semantics, except stable and stage semantics, are based on admissible sets and thus these algorithms enumerate admissible sets in order to construct the respective extensions. To explore the efficiency of these algorithms we implemented them and profiled their performance against solving the encoding of

Table 4.8: The average elapsed time of algorithm 17 versus ASPARTIX, argument systems were generated by setting attacks with a specific probability.

$\|A\|$	probability=$\frac{2\times log_e\|A\|}{\|A\|}$		probability=$\frac{4\times log_e\|A\|}{\|A\|}$	
	ASPARTIX	Algorithm 17	ASPARTIX	Algorithm 17
36	167.60	2.00	242.00	1.70
37	207.10	0.20	273.70	3.20
38	186.40	0.30	300.00	4.80
39	268.90	4.30	339.30	6.00
40	277.60	3.80	354.00	2.00
41	372.20	7.90	403.00	0.50
42	336.50	8.10	442.60	0.30
43	371.30	10.80	504.50	0.60
44	375.80	11.50	558.10	0.20
45	464.30	11.50	594.50	0.30

Table 4.9: The average elapsed time of algorithm 18, argument systems were generated by using algorithm 11.

$\|A\|$	Range of $\|R\|$	algorithm 18
51	75-1484	10.80
52	52-1456	9.00
53	78-1519	10.60
54	54-1698	11.40
55	55-1704	13.10
56	56-1769	15.80
57	57-1892	17.90
58	5-1788	18.50
59	59-1848	17.20
60	60-1929	21.40

ASPARTIX [56] using DLV system [69]. One main objective of this chapter was to show whether heuristics and pruning strategies exploited in enumerating preferred extensions are equally useful for deciding extensions under other semantics. Therefore, we found that heuristics and pruning strategies applied for preferred semantics are totally transferable to complete, ideal and semi stable semantics. We have shown, however, that stage and stable semantics need slightly different approaches concerning heuristics and pruning tactics. In fact, the existing algorithms of [35, 78] can also be re-engineered towards developing algorithms for the semantics addressed in this chapter. For example, [25, 27] presented algorithms for enumerating semi stable, respectively stage, extensions based on the algorithm of [78]. [41] (respectively [48]) presented an algorithm that constructs the ideal extension given that the credulous

Algorithm 19: Deciding the grounded extension of an argument system (A, R).

```
1 Lab : A → {IN, OUT, UNDEC}; Lab ← φ;
2 foreach w ∈ A do Lab ← Lab ∪ {(w, UNDEC)};
3 while ∃x ∈ A : Lab(x) = UNDEC ∧ ∀y ∈ {x}⁻ (Lab(y) = OUT) do
4     foreach x ∈ A : Lab(x) = UNDEC ∧ ∀y ∈ {x}⁻ (Lab(y) = OUT) do
5         Lab(x) ← IN;
6         foreach z ∈ {x}⁺ do Lab(z) ← OUT;
7 report the grounded extension is {w | Lab(w) = IN};
```

Table 4.10: The average elapsed time of algorithm 19 versus ASPARTIX, argument systems were generated by using algorithm 11.

$\|A\|$	Range of $\|R\|$	ASPARTIX	algorithm 19
110	110-6417	216.50	6.80
120	120-7082	264.70	7.20
130	512-9099	286.30	8.10
140	392-10040	378.50	8.20
150	150-11641	468.50	9.00
160	160-13165	591.00	10.80
170	170-14960	633.10	11.30
180	274-16563	813.80	12.40
190	190-18691	961.20	15.00
200	298-21091	1,027.20	14.70

acceptance (respectively the skeptical acceptance) for each argument is already identified. [33] presented a significant algorithm for enumerating extensions of the theory of [91] for default reasoning, which can be also used for computing stable extensions. The algorithm of [33] uses four labels that correspond to: IN, OUT, UNDEC and "FORBIDDEN". In contrast to algorithm 14 for stable semantics, the FORBIDDEN label is used instead of the labels IGNORED and MUST_OUT. In sections 3.2 & 3.5 we illustrated the advantages of using the labels: IGNORED and MUST_OUT. The heuristics and pruning strategies of [33] intersect with the approaches of algorithm 14. Highlighting differences, we explained in section 4.1 that the IGNORED label allows for an additional pruning property to be exploited. As to the rules for selecting the next argument to be labeled IN, the algorithm of [33] applied further two advanced criteria in addition to the measures that are adopted by algorithm 14. Showing one of these advanced criteria, [33] might select an argument from the minimal set $S \subseteq A$ that makes $\{(x, y) \in R : x, y \in A \setminus S\}$ acyclic. However, [33] stated that finding such a set S is likely to be an intractable problem, and hence [33] uses an approximation method to compute such S. Recall that algorithm 14 adheres to a simple heuristic selection rule. In section 3.3 we have shown, experimentally in terms of overall running times, that

Table 4.11: The average elapsed time of algorithm 19 versus ASPARTIX, argument systems were generated by setting attacks with a specific probability.

$\|A\|$	probability=$\frac{2\times log_e\|A\|}{\|A\|}$		probability=$\frac{4\times log_e\|A\|}{\|A\|}$	
	ASPARTIX	Algorithm 19	ASPARTIX	Algorithm 19
110	173.20	0.30	165.20	0.20
120	216.40	0.00	213.80	0.20
130	272.50	0.00	268.70	0.00
140	338.10	0.00	330.20	0.30
150	406.60	0.60	400.30	0.70
160	494.60	0.50	481.70	0.10
170	592.50	0.40	578.60	0.50
180	698.30	0.50	682.80	0.30
190	819.80	0.00	795.80	0.20
200	950.60	0.20	920.50	0.40

using unsophisticated heuristics tends to be more cost-effective than using involved heuristics as is the case in the algorithm of Doutre and Mengin [35].

Chapter 5

Labeling Algorithms for Value Based Argument Systems

In this chapter we develop algorithms for value based argument systems under preferred semantics. We refer the reader to section 1.5 for the technical background on value based argument systems. In section 5.1 we build algorithms for enumerating preferred extensions. In section 5.2 we verify the efficiency of these algorithms experimentally. In section 5.3 we design algorithms for deciding objective/subjective acceptance without requiring enumeration of all preferred extensions explicitly. Section 5.4 closes the chapter with a summary.

5.1 A Novel Algorithm for Enumerating Preferred Extensions

To determine all preferred extensions over all specific audiences a naive approach would enumerate all specific audiences, leading to $|V|!$ running time. We develop a new approach that avoids forming *all* such audiences leading to improved expected running time. The new approach is presented by algorithms 20, 21, 22, and 23.

Algorithm 20 builds total orders on V (that is, specific audiences) dynamically on the fly. For this purpose, we define $q : V \to \mathbb{Z}$ a mapping from social values to integers[1]. Every time a social value is mapped to an integer by q (line 13), algorithm 20 might call (line 17) algorithm 21 and attempt to label an argument x IN: the effect of the value order encoded in q. In doing so, algorithm 21 may then call (line 10) algorithm 22. Algorithm 22 checks whether an argument, $y \in \{x\}^-$, labelled UNDEC may be labelled OUT under q or not. That is to say whether, w.r.t. the audience described by q, y is defeated. Thus, algorithm 22 might call (line 5) algorithm 21 to decide whether an UNDEC labelled attacker of y can be labelled IN or not. To avoid infinite recursion the algorithms 21 and 22 employ $W \subseteq A$ to hold processed arguments. In summary, every time q is changed, algorithms 21 and 22 together determine IN/OUT labels on UNDEC arguments to reflect this change in q. Once no eligible social value is left

[1] We discuss the benefit of using q with more examples later in this section.

unmapped by q (line 19 of algorithm 20), algorithm 20 calls (line 20) algorithm 23 to find the preferred extensions under the audience encoded by q. Algorithm 23 is almost identical to algorithm 9 with one exception related to the defeat notion of value based argument systems. This requires us to define a transition rule, *IN-TRANS-VAL*, for algorithm 23 instead of IN-TRANS that is used by algorithm 9.

Definition 13. *Let (A, R, V, η) be a value based argument system, $x \in A$, $Lab : A \rightarrow \{IN, OUT, MUST\_OUT, IGNORED, UNDEC\}$ be a total mapping such that $Lab(x) = UNDEC$, and $q : V \rightarrow \mathbb{Z}$. Then the in transition step on x under q, denoted as IN-TRANS-VAL(x, q), is defined by the following steps:*

1. $Lab' \leftarrow Lab$,
2. $Lab'(x) \leftarrow IN$,
3. ***foreach*** $(x, y) \in R : q(\eta(x)) \leq q(\eta(y))$ ***do*** $Lab'(y) \leftarrow OUT$,
4. ***foreach*** $(z, x) \in R : Lab'(z) \neq OUT \wedge q(\eta(z)) \leq q(\eta(x))$ ***do*** $Lab'(z) \leftarrow MUST\_OUT$,
5. ***return*** Lab'.

Algorithm 20: Enumerating all preferred extensions of a value based argument system $H = (A, R, V, \eta)$.

```
1  Lab : A → {IN, OUT, MUST_OUT, IGNORED, UNDEC};
2  Lab ← φ;
3  foreach x ∈ A do
4      Lab ← Lab ∪ {(x, UNDEC)};
5  q : V → ℤ; q ← ∅;
6  foreach v ∈ V do
7      q ← q ∪ {(v, ∞)};
8  i ← 1;
9  call find-preferred-extensions(Lab, q, i);
10 procedure find-preferred-extensions(Lab, q, i)
11 foreach v ∈ V : (q(v) = ∞) ∧ (∃x Lab(x) = UNDEC) ∧ (η(x) = v) do
12     q' ← q;
13     q'(v) ← i;
14     Lab' ← Lab;
15     foreach z : Lab'(z) = UNDEC ∧ η(z) = v do
16         W ← ∅;
17         invoke algorithm 21 on (H, Lab', z, q', W);
18     call find-preferred-extensions(Lab', q', i + 1);
19 if ∄v ∈ V : (q(v) = ∞) ∧ (∃x Lab(x) = UNDEC) ∧ (η(x) = v) then
20     invoke algorithm 23 on (H, Lab, q);
21 end procedure
```

Algorithm 21: Labeling an argument x IN in a value based argument system $H = (A, R, V, \eta)$ under $q : V \to \mathbb{Z}$ given that $Lab : A \to \{IN, OUT, MUST\_OUT, IGNORED, UNDEC\}$ holds the current labels for all arguments and $W \subseteq A$ holds preprocessed arguments.

```
1  W ← W ∪ {x};
2  foreach y ∈ {x}^− : q(η(y)) ≤ q(η(x)) do
3      if Lab(y) = IN then
4          return false;
5      else
6          if y ∈ W ∧ Lab(y) = UNDEC then
7              return false;
8          if y ∉ W ∧ Lab(y) = UNDEC then
9              W' ← W;
10             invoke algorithm 22 on (H, Lab, q, W', y);
11             if algorithm 22 returned false then
12                 return false;
13 Lab(x) ← IN;
14 foreach z ∈ {x}^+ : q(η(x)) ≤ q(η(z)) do
15     Lab(z) ← OUT;
16 return true;
```

Algorithm 22: Labeling an argument y OUT in a value based argument system $H = (A, R, V, \eta)$ under $q : V \to \mathbb{Z}$ given that $Lab : A \to \{IN, OUT, MUST\_OUT, IGNORED, UNDEC\}$ holds the current labels for all arguments and $W \subseteq A$ holds preprocessed arguments.

```
1  W ← W ∪ {y};
2  foreach s ∈ {y}^− : q(η(s)) ≤ q(η(y)) do
3      if s ∉ W ∧ Lab(s) = UNDEC then
4          W' ← W;
5          invoke algorithm 21 on (H, Lab, q, W', s);
6          if algorithm 21 returned true then
7              Lab(y) ← OUT;
8              return true;
9  return false;
```

Algorithm 23: Enumerating preferred extensions of a value based argument system $H = (A, R, V, \eta)$ under $q : V \rightarrow \mathbb{Z}$ given that $Lab : A \rightarrow \{IN, OUT, MUST\_OUT, IGNORED, UNDEC\}$ holds the current labels for all arguments.

```
1  PEXT ← ∅;
2  call find-preferred-extensions(H, Lab);
3  report PEXT is the set of preferred extensions under the audience encoded by q;
4  procedure find-preferred-extensions(H, Lab)
5  if ∀x Lab(x) ≠ UNDEC then
6      if ∀x Lab(x) ≠ MUST_OUT then
7          S ← {y ∈ A | Lab(y) = IN};
8          if ∄T ∈ PEXT : S ⊆ T then
9              PEXT ← PEXT ∪ {S};
10 else
11     select any x ∈ A with Lab(x) = UNDEC;
12     Lab' ← IN − TRANS − VAL(x, q);
13     call find-preferred-extensions(H, Lab');
14     Lab' ← IGNORE − TRANS(x);
15     call find-preferred-extensions(H, Lab');
16 end procedure
```

Figure 5.1 shows how the algorithms work on the framework of figure 1.3. A benefit of $q : V \rightarrow \mathbb{Z}$ defined in algorithm 20, we believe, is that building value orders on V incrementally by using q improves the efficiency of computing preferred extensions. To see why, notice that q does not map the social values that are promoted by OUT labelled arguments because these values are irrelevant in deciding the labels of the remaining UNDEC arguments. Since the example of figure 5.1 does not show the gain of q we present figure 5.2 that shows how the algorithms work on another value based argument system. In this example the algorithms decide the α-preferred extensions in four stages corresponding to q as:

$$\{(v_1, 1), (v_2, \infty), (v_3, 2)\},$$
$$\{(v_1, 2), (v_2, 1), (v_3, \infty)\},$$
$$\{(v_1, 2), (v_2, \infty), (v_3, 1)\},$$
$$\{(v_1, 3), (v_2, 2), (v_3, 1)\}.$$

In contrast, working on a precomputed set of all specific audiences enforces 6 total orders which necessitates 6 stages. More specifically, referring to figure 5.2 the total orders: $v_1 \succ v_2 \succ v_3$ and $v_1 \succ v_3 \succ v_2$ would produce the same α-preferred extensions. Noting that as v_1 is the most preferred value then the argument y is OUT and hence the relative position of v_2 is unimportant. Thus, the value order $v_1 \succ v_2 \succ v_3$ is not critical and can be ignored. This is exactly what our approach does where algorithm 20 does not build a function q to embody $v_1 \succ v_2 \succ v_3$. Similarly, algorithm 20 does not develop a function q to represent the value order $v_2 \succ v_3 \succ v_1$ indicating that the order of v_3 is not

critical: v_3 is associated with only OUT arguments (the argument z in this example).

We now turn to proving the correctness of our approach. Let us first define the set of *critical* total value orders $\mathcal{U}'$. Subsequently we prove that $\mathcal{U}'$ would produce the same α-preferred extensions as those generated by considering all specific audiences. Afterwards, we establish the correctness of algorithms 20, 21, 22 and 23.

Definition 14. *Let $H = (A, R, V, \eta)$ be a value based argument system and $\mathcal{U}$ denote the set of all specific audiences over V. Then the set of* critical *total value orders, $\mathcal{U}'$, is*

$$\begin{aligned}\mathcal{U} \setminus \quad & \{\alpha \in \mathcal{U} \mid \exists (v_i, v_j) \in \alpha \ : v_i \neq v_j \wedge \\ & \forall x \ : \ \eta(x) = v_i, \nexists S_1 \subseteq A \ : S_1 \text{ is admissible under } \alpha \text{ with } x \in S_1 \wedge \\ & \exists S_2 \subseteq A : S_2 \text{ is admissible under } \alpha \wedge \\ & \exists y \in S_2 \ : \ y \text{ defeats } x \text{ w.r.t. } \alpha\}.\end{aligned}$$

Proposition 8. *Let (A, R, V, η) be a value based argument system, PREF be the set of all preferred extensions w.r.t. all specific audiences in $\mathcal{U}$, PREF′ be the set of all preferred extensions w.r.t. the set of critical total value orders $\mathcal{U}'$. Then PREF = PREF′.*

Proof: Assume there exists $T \in PREF : T \notin PREF'$. In this case

$$\begin{aligned}&\exists p_1 \in \mathcal{U} : \ T \text{ is a preferred extension under } p_1 \text{ and} \\ &\forall p_2 \in \mathcal{U}', \ T \text{ is not a preferred extension under } p_2.\end{aligned}$$

By definition of $\mathcal{U}$ and $\mathcal{U}'$

$$\begin{aligned}&\exists (v_i, v_j) \in p_1 : v_i \neq v_j \wedge \\ &\forall x \in A : \eta(x) = v_i, \ \nexists S_1 \subseteq A : S_1 \text{ is admissible under } p_1 \text{ with } x \in S_1 \wedge \\ &\exists S_2 \subseteq A : S_2 \text{ is admissible under } p_1 \wedge \\ &\exists y \in S_2 : y \text{ defeats } x \text{ w.r.t. } p_1.\end{aligned}$$

Thus,

$$\begin{aligned}&\forall p_2 \in \mathcal{U}' : p_2 = \\ &\quad (p_1 \setminus \{(v_i, v_j) \in p_1 \mid v_i \neq v_j \wedge \\ &\quad \forall x \in A : \eta(x) = v_i, \ \nexists S_1 \subseteq A : S_1 \text{ is admissible under } p_1 \text{ with } x \in S_1 \wedge \\ &\quad \exists S_2 \subseteq A : S_2 \text{ is admissible under } p_1 \text{ with some } y \in S_2 : y \text{ defeats } x \text{ w.r.t. } p_1\}) \\ &\quad \cup \\ &\quad \{(v_k, v_l) \notin p_1 \mid (v_l, v_k) \in p_1 \wedge \\ &\quad \forall x \in A : \eta(x) = v_l, \ \nexists S_1 \subseteq A : S_1 \text{ is admissible under } p_1 \text{ with } x \in S_1 \wedge \\ &\quad \exists S_2 \subseteq A : S_2 \text{ is admissible under } p_1 \text{ with some } y \in S_2 : y \text{ defeats } x \text{ w.r.t. } p_1\},\end{aligned}$$

$$\forall T \in PREF : \ T \text{ is a preferred extension under } p_1, \ T \text{ is a preferred extension under } p_2.$$

Contradiction. ∎

Proposition 9. *Let (A, R, V, η) be a value based argument system and $\mathcal{U}'$ be the set of all critical total value orders. Then:*

1. *for every $q : V \to \mathbb{Z}$ constructed by algorithm 20, $\exists p \in \mathcal{U}' : \forall v_i, v_j \in V : \ q(v_i) \leq q(v_j), (v_i, v_j) \in p$.*

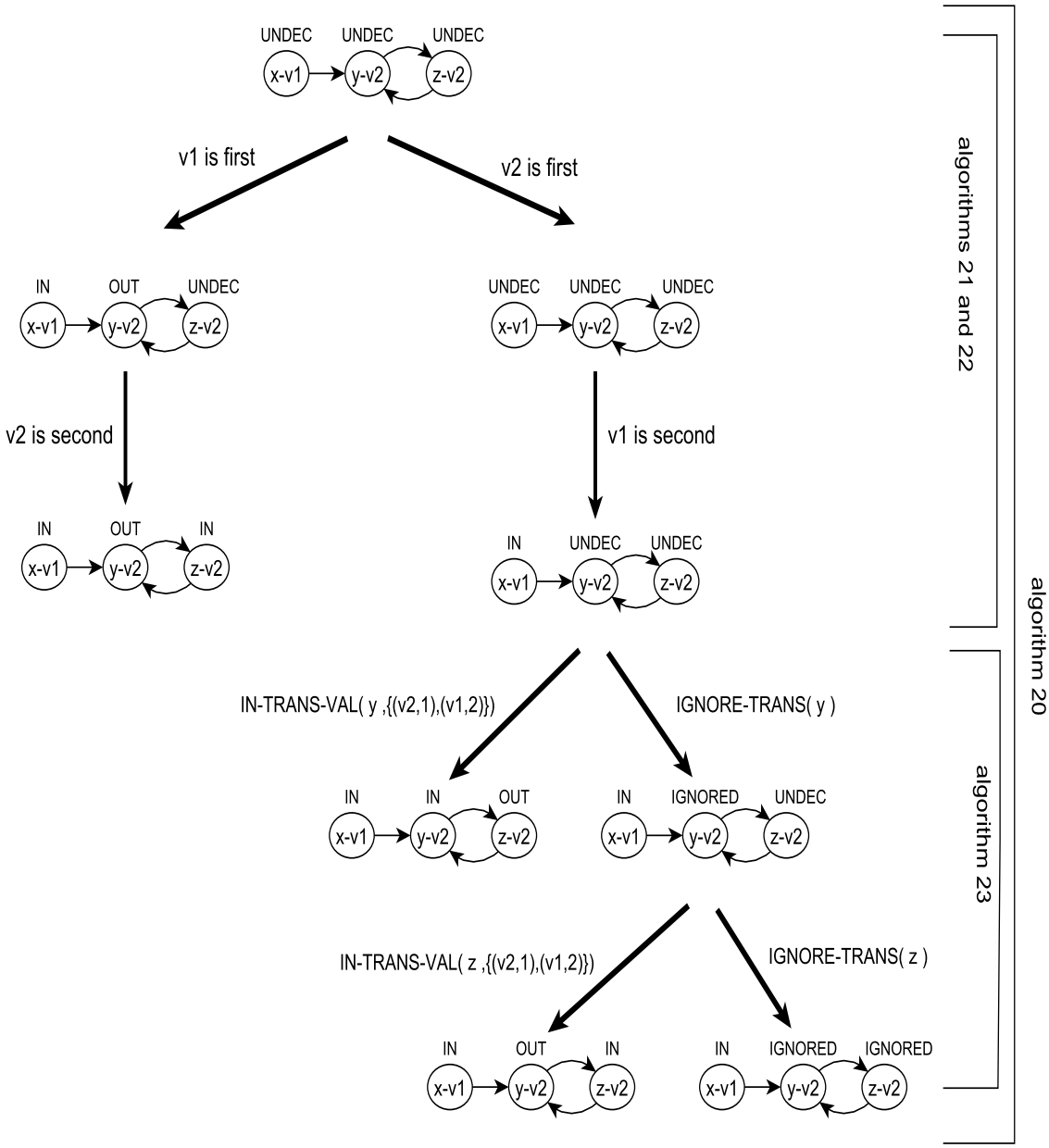

Figure 5.1: How algorithms 20, 21, 22 and 23 work on a value based argument system.

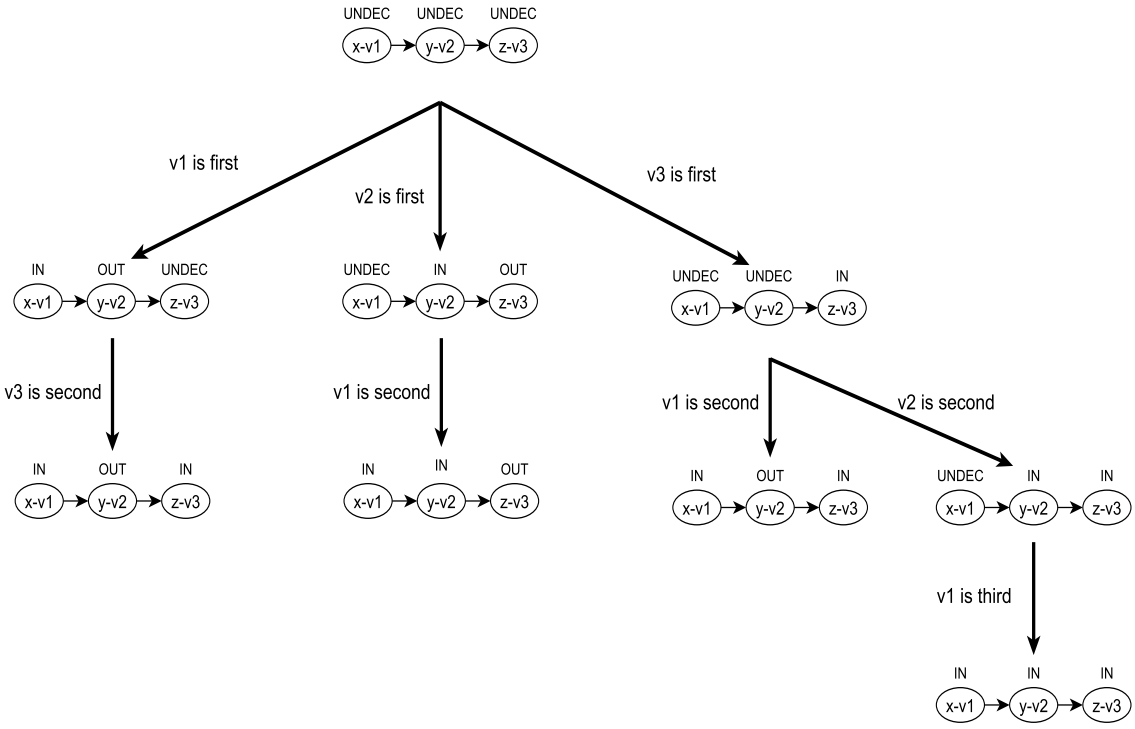

Figure 5.2: Showing the benefit of our approach by tracing algorithms 20, 21, 22 and 23 on a value based argument system.

2. $\forall p \in \mathcal{U}'$, *algorithm 20 constructs* $q : V \rightarrow \mathbb{Z}$ *such that* $\forall v_i, v_j \in V : (v_i, v_j) \in p,\ q(v_i) \leq q(v_j)$.

Proof:

1. Assume

   $$\exists q \textit{ constructed by algorithm } 20 : \forall p \in \mathcal{U}',\ \exists v_i, v_j \in V : q(v_i) \leq q(v_j) \wedge (v_i, v_j) \notin p.$$

   By Algorithm 20, lines 15 and 17

   $$\exists x \in A : \eta(x) = v_i \textit{ and } Lab(x) \in \{IN, UNDEC\} \textit{ w.r.t. } q.$$

   By proposition 10 (of which paragraph 1 states the conditions under which an argument is labelled IN or left UNDEC by algorithm 21) and after simplifications

   $$\exists S_1 \subseteq A : S_1 \textit{ is admissible under } q \textit{ with } x \in S_1 \textit{ or}$$
   $$\nexists S_2 \subseteq A : S_2 \textit{ is admissible under } q \textit{ with some } y \in S_2 \cap \{x\}^- \textit{ s.t. } q(\eta(y)) \leq q(\eta(x)).$$

   This contradicts the definition of $\mathcal{U}'$ (definition 14): recall that $\mathcal{U}'$ contains total value orders including those which satisfy the previous implication.

2. Assume that

   $$\exists p \in \mathcal{U}' : \textit{for every } q : V \rightarrow \mathbb{Z} \textit{ constructed by algorithm } 20,$$
   $$\exists v_i, v_j \in V : (v_i, v_j) \in p \wedge q(v_i) > q(v_j).$$

By definition of $\mathcal{U}'$

$$\exists x \in A : \eta(x) = v_i,\ \exists S_1 \subseteq A : S_1 \textit{ is admissible under } p \textit{ with } x \in S_1 \textit{or}$$
$$\nexists S_2 \subseteq A : S_2 \textit{ is admissible under } p \textit{ with some } y \in S_2 : y \textit{ defeats } x \textit{ w.r.t. } p.$$

By proposition 10, x will be labelled IN or it stays UNDEC.
By algorithm 20 (lines 8, 11 and 13), there exists $q(v_i) = 1$. Contradiction.

■

Proposition 10. *Let (A, R, V, η) be a value based argument system, $Lab(x) = UNDEC$ and $q : V \to \mathbb{Z}$ formed by algorithm 20. Then:*

1. *$Lab(x) \leftarrow IN$ under q by algorithm 21 if and only if*

$$\exists S_1 \subseteq A : S_1 \textit{ is admissible under } q \textit{ with } x \in S_1 \textit{ and}$$
$$\nexists S_2 \subseteq A : S_2 \textit{ is admissible under } q \textit{ with some } y \in S_2 \cap \{x\}^- \textit{ s.t. } q(\eta(y)) \leq q(\eta(x)).$$

2. *$Lab(x) \leftarrow OUT$ under q by algorithm 21 (respectively algorithm 22) if and only if*

$$\nexists S_1 \subseteq A : S_1 \textit{ is admissible under } q \textit{ with } x \in S_1 \textit{ and}$$
$$\exists S_2 \subseteq A : S_2 \textit{ is admissible under } q \textit{ with some } y \in S_2 \cap \{x\}^- \textit{ s.t. } q(\eta(y)) \leq q(\eta(x)).$$

3. *If neither (1) nor (2) hold, $Lab(x) = UNDEC$.*

Proof:

1. Assume x is labelled IN under q by algorithm 21 and

$$\nexists S_1 \subseteq A : S_1 \textit{ is admissible under } q \textit{ with } x \in S_1 \textit{ or}$$
$$\exists S_2 \subseteq A : S_2 \textit{ is admissible under } q \textit{ with some } y \in S_2 \cap \{x\}^- \textit{ s.t. } q(\eta(y)) \leq q(\eta(x)).$$

   By algorithm 21 lines 4, 7 & 12

$$\forall (y, x) \in R : q(\eta(y)) \leq q(\eta(x))\ Lab(y) = OUT.\ \texttt{Contradiction.}$$

2. Assume x is labelled OUT under q by algorithm 21 (respectively algorithm 22) and

$$\exists S_1 \subseteq A : S_1 \textit{ is admissible under } q \textit{ with } x \in S_1 \textit{ or}$$
$$\nexists S_2 \subseteq A : S_2 \textit{ is admissible under } q \textit{ with some } y \in S_2 \cap \{x\}^- \textit{ s.t. } q(\eta(y)) \leq q(\eta(x)).$$

   By algorithm 21 line 15 (respectively algorithm 22 line 7)

$$\exists (y, x) \in R : q(\eta(y)) \leq q(\eta(x)) \wedge Lab(y) = IN.\ \texttt{Contradiction}.$$

3. Immediate from (1) & (2).

■

Proposition 11. *Let (A, R, V, η) be a value based argument system, $q : V \rightarrow \mathbb{Z}$, and let PEXT be the preferred extensions under q decided by algorithm 23. Then,*

1. *$\forall pext \in PEXT$, $\exists S \subseteq A : S$ is a preferred extension under $q \wedge S = pext$, and*
2. *$\forall S \subseteq A$, if S is a preferred extension under q then $\exists pext \in PEXT : S = pext$.*

Proof: Follows from proposition 6 and similar structure of algorithm 23 and algorithm 10. ∎

We close this section by considering rewriting a value based argument system into a Dung argument system [76, 77] by, in general, adding an argument to A for every specific audience $p \in \mathcal{U}$; thus A will grow by $|V|!$. So, any algorithm (e.g. algorithm 9) working on the target argument system will run in the order of $2^{|A|+|V|!}$ while our approach (i.e. algorithms 20, 21, 22 and 23 altogether working on the original value based argument system) runs in the order of $|V|! * 2^{|A|}$ which is more efficient. More importantly, recall the profit of q that might induce steps fewer than $|V|!$ as we illustrated earlier. The bottom line is, our approach is faster than any other mechanism that would consider every specific audience $p \in \mathcal{U}$ where our algorithms only consider critical audiences $\mathcal{U}' \subseteq \mathcal{U}$ as stated by proposition 9, which is a key contribution of our algorithms. Equally, our algorithms establish an efficient method for encoding total orders over V such that the space complexity is upper bounded to the squared number of values (i.e. $|V|^2$) rather than to the number of all total value orders (i.e. $|V!|$), which is the case if a naive approach is adopted.

5.2 Experimental Evaluation

We present experimental results to support our claim in the last section regarding the benefit of our algorithms. The implementation was in C++, on a Fedora (release 13) based machine with 4 processors (Intel core i5-750 2.67GHz) and 16GB of memory. We tested the algorithms of this chapter with 25,000 randomly generated instances. We generated instances of value based argument systems by setting attacks between arguments with a probability of 0.1 where each instance has 30 arguments. We report in table 5.1 the average number of processed total value orders in an execution of 100 instances.

Table 5.1: The average number of total value orders processed in executing algorithms 20, 21, 22 and 23.

$\|V\|$	5	6	7	8	9
naive approach	120.00	720.00	5,040.00	40,320.00	362,880.00
new approach	115.88	650.34	3,843.91	25,653.87	171,224.62

5.3 Algorithms for Subjective and Objective Acceptance

Here we provide algorithms that decide subjective/objective acceptance without explicitly enumerating extensions of a value based argument system. Algorithms 24 and 25 (besides algorithms 21 and 22) decide subjective acceptance while the algorithms 26 and 27 (besides algorithms 21 and 22) decide objective acceptance. In fact these algorithms are self-explanatory since they are alterations of algorithms 12, 13 and 20 as we specify in what follows. Firstly, note that algorithms 24 and 26 are modified versions of algorithm 20 for deciding the subjective, respectively the objective, acceptance problem. Secondly, we reform algorithm 12 to get algorithm 25 that works in conjunction with algorithm 24 to decide the subjective acceptance problem. Thirdly, we change algorithm 13 to get algorithm 27 that works jointly with algorithm 26 in deciding the objective acceptance problem.

Algorithm 24: Deciding subjective acceptance of x in a value based argument system $H = (A, R, V, \eta)$.

$Lab : A \rightarrow \{IN, OUT, MUST\_OUT, IGNORED, UNDEC\}$; $Lab \leftarrow \phi$;
foreach $y \in A$ **do** $Lab \leftarrow Lab \cup \{(y, UNDEC)\}$;
$q : V \rightarrow \mathbb{Z}$; $q \leftarrow \phi$;
foreach $v \in V$ **do** $q \leftarrow q \cup \{(v, \infty)\}$;
$i \leftarrow 1$;
if is-subjectively-accepted(Lab, q, i) **then** x is subjectively accepted;
else x is not subjectively accepted;
procedure is-subjectively-accepted(Lab, q, i)
foreach $v \in V : (q(v) = \infty) \wedge (\exists y : Lab(y) = UNDEC) \wedge (\eta(y) = v)$ **do**
    $q' \leftarrow q$;
    $q'(v) \leftarrow i$;
    $Lab' \leftarrow Lab$;
    foreach $z : Lab'(z) = UNDEC \wedge \eta(z) = v$ **do**
        $W \leftarrow \emptyset$;
        invoke algorithm 21 with (Lab', H, z, q', W);
        if $Lab'(x) = IN$ **then**
            return true;
    if *is-subjectively-accepted*($Lab', q', i + 1$) **then**
        return true;
if $Lab(x) = UNDEC$ *and* $\nexists v \in V : (q(v) = \infty) \wedge (\exists y : Lab(y) = UNDEC) \wedge (\eta(y) = v)$ **then**
    invoke algorithm 25 passing on (H, x, q);
    if *algorithm 25 reports x is credulously accepted* **then**
        return true;
return false;
end procedure

Algorithm 25: Finding a credulous proof of x in a value based argument system $H = (A, R, V, \eta)$ w.r.t. $q : V \to \mathbb{Z}$.

1 $Lab : A \to \{PRO, OPP, OUT, MUST\_OUT, IGNORED, UNDEC\}$; $Lab \leftarrow \phi$;
2 **foreach** $y \in A$ **do** $Lab \leftarrow Lab \cup \{(y, UNDEC)\}$;
3 $Lab(x) \leftarrow PRO$;
4 **foreach** $(x, y) \in R : q(\eta(x)) \leq q(\eta(y))$ **do** $Lab(y) \leftarrow OUT$;
5 **foreach** $(z, x) \in R : q(\eta(z)) \leq q(\eta(x))$ **do**
6 **if** $Lab(z) \in \{IGNORED, UNDEC\}$ **then**
7 $Lab(z) \leftarrow MUST\_OUT$;
8 **if** $\nexists (w, z) \in R : Lab(w) = UNDEC$ *and* $q(\eta(w)) \leq q(\eta(z))$ **then**
9 x is not credulously accepted; **exit**;
10 **else**
11 **if** $Lab(z) = OUT$ **then**
12 $Lab(z) \leftarrow OPP$;
13 **if** *is-accepted(Lab)* **then**
14 x is proved by $\{y \in A \mid Lab(y) \in \{PRO, OPP\}\}$;
15 **else**
16 x is not credulously acceptable;
17 **procedure** is-accepted(Lab)
18 **foreach** $y \in A : Lab(y) = MUST\_OUT$ **do**
19 **while** $\exists (z, y) \in R : Lab(z) = UNDEC \wedge q(\eta(z)) \leq q(\eta(y))$ **do**
20 **select** $z \in \{y\}^-$ s.t. $Lab(z) = UNDEC$ and $q(\eta(z)) \leq q(\eta(y))$ and $\forall w \in \{z\}^- : q(\eta(w)) \leq q(\eta(z))$ $(Lab(w) \in \{OPP, OUT, MUST\_OUT\})$, **otherwise select** $z \in \{y\}^-$ s.t. $Lab(z) = UNDEC$ and $q(\eta(z)) \leq q(\eta(y))$ and $\forall w \in \{y\}^- : q(\eta(w)) \leq q(\eta(y)) \wedge Lab(w) = UNDEC$ $(|\{s \in \{z\}^+ : q(\eta(z)) \leq q(\eta(s))\}| \geq |\{t \in \{w\}^+ : q(\eta(w)) \leq q(\eta(t))\}|)$;
21 $Lab' \leftarrow Lab$; $Lab'(z) \leftarrow PRO$;
22 **foreach** $(z, u) \in R : q(\eta(z)) \leq q(\eta(u))$ **do**
23 **if** $Lab'(u) = MUST\_OUT$ **then**
24 $Lab'(u) \leftarrow OPP$;
25 **else**
26 **if** $Lab'(u) \neq OPP$ **do** $Lab'(u) \leftarrow OUT$;
27 **foreach** $(u, z) \in R : q(\eta(u)) \leq q(\eta(z))$ **do**
28 **if** $Lab'(u) \in \{IGNORED, UNDEC\}$ **then**
29 $Lab'(u) \leftarrow MUST\_OUT$;
30 **if** $\nexists (w, u) \in R : Lab'(w) = UNDEC$ *and* $q(\eta(w)) \leq q(\eta(u))$ **then**
31 $Lab(z) \leftarrow IGNORED$; goto line 19;
32 **else**
33 **if** $Lab'(u) = OUT$ **then**
34 $Lab'(u) \leftarrow OPP$;
35 **if** *is-accepted(Lab')* **then**
36 $Lab \leftarrow Lab'$; return true;
37 **else**
38 $Lab(z) \leftarrow IGNORED$;
39 return false;
40 return true;
41 **end procedure**

Algorithm 26: Deciding objective acceptance of x in a value based argument system $H = (A, R, V, \eta)$.

```
1  Lab : A → {IN, OUT, MUST_OUT, IGNORED, UNDEC};  Lab ← φ;
2  foreach y ∈ A do Lab ← Lab ∪ {(y, UNDEC)};
3  q : V → ℤ;
4  q ← φ;
5  foreach  v ∈ V do
6      q ← q ∪ {(v, ∞)};
7  i ← 1;
8  if  is-objectively-accepted(Lab, q, i) then
9      x is objectively accepted;
10 else
11     x is not objectively accepted;
12 procedure is-objectively-accepted(Lab, q, i)
13 foreach v ∈ V : (q(v) = ∞) ∧ (∃y : Lab(y) = UNDEC) ∧ (η(y) = v) do
14     q' ← q;
15     q'(v) ← i;
16     Lab' ← Lab;
17     foreach z : Lab'(z) = UNDEC ∧ η(z) = v do
18         W ← ∅;
19         invoke algorithm 21 (Lab', H, z, q', W);
20         if Lab'(x) = OUT then
21             return false;
22     if ¬ is-objectively-accepted(Lab', q', i + 1) then
23         return false;
24 if Lab(x) = UNDEC and ∄v ∈ V : (q(v) = ∞) ∧ (∃y : Lab(y) = UNDEC) ∧ (η(y) = v)
   then
25     invoke algorithm 27 with (Lab, H, x, q);
26     if algorithm 27 decided that x is not skeptically accepted then
27         return false;
28 return true;
29 end procedure
```

Algorithm 27: Deciding the skeptical acceptance of x in a value based argument system $H = (A, R, V, \eta)$ w.r.t. $q : V \rightarrow \mathbb{Z}$ given a total labelling $Lab : A \rightarrow \{IN, OUT, MUST\_OUT, IGNORED, UNDEC\}$.

1 $PEXT \leftarrow \emptyset$;
2 **if** $\nexists (y, x) \in R : q(\eta(y)) \leq q(\eta(x))$ **then**
3 x is skeptically proved by $\{x\}$; exit;
4 **foreach** $(y, x) \in R : q(\eta(y)) \leq q(\eta(x))$ **do**
5 **invoke** algorithm 25 on (H, y, q);
6 **if** *algorithm 25 decided that y is credulously accepted* **then**
7 x is not skeptically accepted; exit;
8 **call** decide-skeptical-acceptance(Lab);
9 **if** $PEXT \neq \emptyset$ **then**
10 x is skeptically proved by $PEXT$; exit;
11 **procedure** decide-skeptical-acceptance(Lab)
12 **while** $\exists y : Lab(y) = UNDEC$ **do**
13 **select** y s.t. $Lab(y) = UNDEC$ and $\forall (z, y) \in R : q(\eta(z)) \leq q(\eta(y))$ $(Lab(z) \in \{OUT, MUST\_OUT\})$,
14 **otherwise select** y s.t. $Lab(y) = UNDEC$ and $\forall w : Lab(w) = UNDEC$ $(|\{s \in \{y\}^+ : q(\eta(y)) \leq q(\eta(s))\}| \geq |\{t \in \{w\}^+ : q(\eta(w)) \leq q(\eta(t))\}|)$;
15 $Lab' \leftarrow Lab$; $Lab'(y) \leftarrow IN$;
16 **foreach** $(y, z) \in R : q(\eta(y)) \leq q(\eta(z))$ **do**
17 $Lab'(z) \leftarrow OUT$;
18 **foreach** $(z, y) \in R : q(\eta(z)) \leq q(\eta(y))$ **do**
19 **if** $Lab'(z) \in \{IGNORED, UNDEC\}$ **then**
20 $Lab'(z) \leftarrow MUST\_OUT$;
21 **if** $\nexists (w, z) \in R : Lab'(w) = UNDEC \wedge q(\eta(w)) \leq q(\eta(z))$ **then**
22 $Lab(y) \leftarrow IGNORED$; goto line 12;
23 **call** decide-skeptical-acceptance(Lab');
24 **if** $\exists (z, y) \in R : Lab(z) \in \{IGNORED, UNDEC\}$ **then**
25 $Lab(y) \leftarrow IGNORED$;
26 **else**
27 $Lab \leftarrow Lab'$;
28 **if** $\nexists y : Lab(y) = MUST\_OUT$ **then**
29 $S \leftarrow \{y \mid Lab(y) = IN\}$;
30 **if** $\nexists T \in PEXT : S \subseteq T$ **then**
31 $PEXT \leftarrow PEXT \cup \{S\}$;
32 **if** $Lab(x) \neq IN$ **then**
33 $PEXT \leftarrow \emptyset$;
34 x is not skeptically accepted; **terminate**;
35 **end procedure**

5.4 Summary

In the context of value based argument systems, we developed novel labeling based algorithms for deciding preferred extensions and subjective/objective acceptance. Highlighting some related works, the computational complexity of subjective/objective

acceptance is believed to be intractable (see e.g. [40, 68, 67]). The algorithms of [42] decide the preferred extensions under the assumption of multi-value cycles (i.e. no cycle includes exclusively arguments that are all mapped to the same social value [13].) Notice that the algorithms presented in chapter 2 for deciding objective/subjective acceptance in value based argument systems are also under the assumption of multi-value cycles. However, the algorithms of this chapter solve an arbitrary value based argument system.

Chapter 6

Labeling Attacks as a Generalization of Labeling Arguments

In this chapter we illustrate how to enumerate extensions under a number of argumentation semantics by labeling attacks along with arguments instead of labeling arguments only. This is of interest in argumentation formalisms that allow attacks on attacks e.g. [76, 57, 7]. To achieve this goal, we develop algorithms for enumerating extensions of argument systems with recursive attacks [7] under preferred, stable, complete, stage, semi stable, grounded and ideal semantics in sections 6.1, 6.2, 6.3, 6.4, 6.5, 6.6 and 6.7 respectively. We refer the reader to section 1.6 for the background on argument systems with recursive attacks. We believe that enumerating extensions by labeling attacks alongside arguments is computationally expensive as much as listing extensions by labeling arguments only. To confirm this, we report in each section an experimental evaluation on the efficiency of the concerned algorithm. Lastly, section 6.8 closes the chapter with a summary.

6.1 Enumerating Preferred Extensions

Algorithm 28 enumerates all preferred extensions. The idea is based on using five labels: IN, OUT, MUST_OUT, IGNORED and UNDEC. An attack $\alpha \in \overline{R}$ is labeled IN to indicate that α might be in a preferred extension. An argument, say x, is labeled OUT if and only if there is $\alpha \in \overline{R}$ *s.t.* α is IN and $trg(\alpha) = x$. An attack $\beta \in \overline{R}$ is labeled OUT if and only if there is $\alpha \in \overline{R}$ *s.t.* α is IN and $trg(\alpha) \in \{\beta, src(\beta)\}$. An argument x is labeled IN, suggesting that x might be in a preferred extension, if and only if:

(i) $\exists \alpha \in \overline{R} : \alpha$ is IN and $src(\alpha) = x$ or

(ii) $\forall \beta \in \overline{R} : trg(\beta) = x$, β is OUT.

An attack β is labeled MUST_OUT if and only if there is $\alpha \in \overline{R}$ *s.t.* α is IN and $trg(\beta) \in \{\alpha, src(\alpha)\}$. An attack α is labeled IGNORED trying to find a preferred extension excluding α. UNDEC is the initial label for all arguments and attacks. Nonetheless, the precise approach to enumerating preferred extensions is defined in algorithm 28. To get the general idea see figure 6.1 that shows how the algorithm works on the system depicted in figure 1.4. Now we highlight heuristics and pruning strategies employed by our algorithm. Referring to algorithm 28, line 9 represents the strategy by which we pick the next attack to be labeled IN. This heuristic rule, and its grounds as well, is in parallel to the heuristic rule applied in algorithm 10 for enumerating preferred extensions of standard argument systems. Likewise, algorithm 28 applies two pruning tactics, which are in line with the pruning strategies of algorithm 10. For the first pruning tactic, algorithm 28 backtracks if there is $\beta \in \overline{R}$ labelled MUST_OUT while there is no $\lambda \in \overline{R}$ is labeled UNDEC with $trg(\lambda) \in \{\beta, src(\beta)\}$, see lines 20- 23. For the second pruning tactic, algorithm 28 (see lines 25- 28) skips ignoring an attack $\alpha \in \overline{R}$ if and only if for each $\beta \in \overline{R}$ *s.t.* $trg(\beta) \in \{\alpha, src(\alpha)\}$, β is OUT or MUST_OUT. Table 6.1 shows the efficiency of algorithm 28 versus algorithm 10 working on argument systems with recursive attacks expressed as standard argument systems [7] as follows: given an argument system with recursive attacks $(A, \overline{R})$, then the corresponding Dung argument system (A', R') is formed with $A' = A \cup \overline{R}$ and $R' = \{(x, y) \mid x, y \in A \cup \overline{R}$ *and* x *defeats* $y\}$. Hence, both algorithms 28 and 10 run in the order of $2^{|A|+|\overline{R}|}$. This is confirmed by the experimental results of table 6.1. Note that the differences between the two algorithms with regards to the elapsed time shown in table 6.1 are negligible. We do not see such subtle differences contradict with the earlier mathematical analysis indicating that the overall performance of both algorithms is comparable.

6.2 Enumerating Stable Extensions

Algorithm 29 enumerates all stable extensions. Actually, algorithm 29 is a modification of algorithm 28, which decides preferred extensions. In algorithm 28 we find a preferred extension, say $pref$, if and only if for each $x \in A \cup \overline{R}$ x is not UNDEC nor MUST_OUT and $pref$ is not a subset of a previously decided preferred extension. However, in algorithm 29 we encounter a stable extension if and only if for each $x \in A \cup \overline{R}$ x is not UNDEC nor MUST_OUT nor IGNORED. Moreover, algorithm 29 applies an additional pruning strategy such that the algorithm backtracks if a condition holds, see lines 26 & 34. Table 6.2 displays the performance of algorithm 29 in contrast to algorithm 14.

Algorithm 28: Enumerating preferred extensions of an argument system with recursive attacks $(A, \overline{R})$.

1 $N(x) \equiv \begin{cases} 0 & \text{if } x \in \overline{R}; \\ |\{\beta \in \overline{R} \mid src(\beta) = x\}| & \text{if } x \in A; \end{cases}$
2 $Lab : (A \cup \overline{R}) \to \{IN, OUT, MUST\_OUT, IGNORED, UNDEC\}$; $Lab \leftarrow \phi$;
3 **foreach** $x \in A \cup \overline{R}$ **do** $Lab \leftarrow Lab \cup \{(x, UNDEC)\}$;
4 $\overline{E}_{preferred} \subseteq 2^{A \cup \overline{R}}$; $\overline{E}_{preferred} \leftarrow \phi$;
5 **call** find-preferred-extensions(Lab);
6 **report** $\overline{E}_{preferred}$ is the set of all preferred extensions;

7 **procedure** find-preferred-extensions(Lab)
8 **while** $\exists \alpha \in \overline{R} : Lab(\alpha) = UNDEC$ **do**
9 select $\alpha \in \overline{R} : Lab(\alpha) = UNDEC$ and
$\forall \beta \in \overline{R} : trg(\beta) \in \{\alpha, src(\alpha)\}$ $(Lab(\beta) \in \{OUT, MUST\_OUT\})$, else pick
$\alpha \in \overline{R} : Lab(\alpha) = UNDEC$ and $Lab(trg(\alpha)) \neq OUT$ and $\forall \beta \in \overline{R} : Lab(\beta) = UNDEC$
$(N(trg(\alpha)) \geq N(trg(\beta)))$, otherwise select any $\alpha \in \overline{R} : Lab(\alpha) = UNDEC$;
10 $Lab' \leftarrow Lab$;
11 $Lab'(\alpha) \leftarrow IN$;
12 $Lab'(src(\alpha)) \leftarrow IN$;
13 **if** $trg(\alpha) \in A$ **then**
14 $Lab'(trg(\alpha)) \leftarrow OUT$;
15 **foreach** $\beta \in \overline{R} : src(\beta) = trg(\alpha)$ **do**
16 $Lab'(\beta) \leftarrow OUT$;
17 **else**
18 $Lab'(trg(\alpha)) \leftarrow OUT$;
19 **foreach** $\beta \in \overline{R} : Lab'(\beta) \in \{UNDEC, IGNORED\} \wedge trg(\beta) \in \{\alpha, src(\alpha)\}$ **do**
20 $Lab'(\beta) \leftarrow MUST\_OUT$;
21 **if** $\nexists\, \lambda \in \overline{R} : Lab'(\lambda) = UNDEC \wedge trg(\lambda) \in \{\beta, src(\beta)\}$ **then**
22 $Lab(\alpha) \leftarrow IGNORED$;
23 **goto** line 8;
24 **call** find-preferred-extensions(Lab');
25 **if** $\exists \beta \in \overline{R} : Lab(\beta) \in \{UNDEC, IGNORED\} \wedge trg(\beta) \in \{\alpha, src(\alpha)\}$ **then**
26 $Lab(\alpha) \leftarrow IGNORED$;
27 **else**
28 $Lab \leftarrow Lab'$;
29 **if** $\nexists \beta \in \overline{R} : Lab(\beta) = MUST\_OUT$ **then**
30 **foreach** $x \in A : Lab(x) = UNDEC \wedge \forall \beta \in \overline{R} : trg(\beta) = x$ $(Lab(\beta) = OUT)$ **do**
31 $Lab(x) \leftarrow IN$;
32 $S \leftarrow \{x \in (A \cup \overline{R}) \mid Lab(x) = IN\}$;
33 **if** $\forall T \in \overline{E}_{preferred}$ $(S \not\subseteq T)$ **then**
34 $\overline{E}_{preferred} \leftarrow \overline{E}_{preferred} \cup \{S\}$;
35 **end procedure**

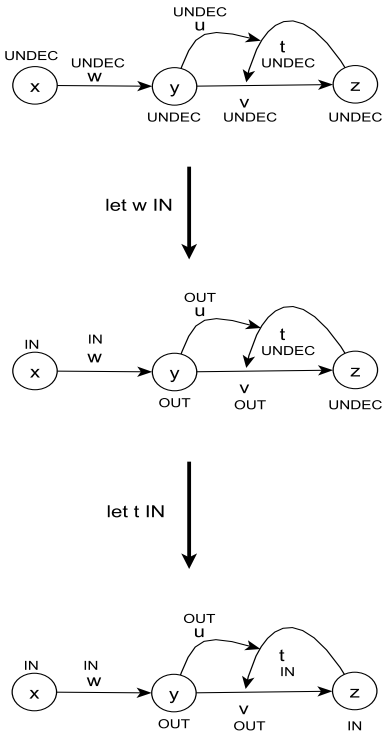

Figure 6.1: How algorithm 28 works on an argument system with recursive attacks.

Table 6.1: The average elapsed time of algorithm 28 versus algorithm 10, instances of argument system with recursive attacks were generated randomly with $|A| = 25$ and by setting attacks with a probability p.

p	0.01	0.02	0.03	0.04	0.05	0.06
average of $\|\overline{R}\|$	7.63	17.73	31.77	46.62	65.64	91.22
algorithm 28	5.40	4.70	0.80	4.30	66.90	1,320.70
algorithm 10	9.50	9.80	10.00	10.50	126.60	2,059.60

Algorithm 29: Enumerating stable extensions of an argument system with recursive attacks $(A, \overline{R})$.

1 $N(x) \equiv \begin{cases} 0 & \text{if } x \in \overline{R}; \\ |\{\beta \in \overline{R} \mid src(\beta) = x\}| & \text{if } x \in A; \end{cases}$
2 $Lab : (A \cup \overline{R}) \rightarrow \{IN, OUT, MUST\_OUT, IGNORED, UNDEC\}$; $Lab \leftarrow \phi$;
3 **foreach** $x \in A \cup \overline{R}$ **do** $Lab \leftarrow Lab \cup \{(x, UNDEC)\}$;
4 $\overline{E}_{stable} \subseteq 2^{A \cup \overline{R}}$; $\overline{E}_{stable} \leftarrow \phi$;
5 **call** find-stable-extensions(Lab);
6 **report** $\overline{E}_{stable}$ is the set of all stable extensions;

7 **procedure** find-stable-extensions(Lab)
8 **while** $\exists \alpha \in \overline{R} : Lab(\alpha) = UNDEC$ **do**
9 select $\alpha \in \overline{R} : Lab(\alpha) = UNDEC$ and
$\forall \beta \in \overline{R} : trg(\beta) \in \{\alpha, src(\alpha)\}$ $(Lab(\beta) \in \{OUT, MUST\_OUT\})$, else pick
$\alpha \in \overline{R} : Lab(\alpha) = UNDEC$ and $Lab(trg(\alpha)) \neq OUT$ and $\forall \beta \in \overline{R} : Lab(\beta) = UNDEC$
$(N(trg(\alpha)) \geq N(trg(\beta)))$, otherwise select any $\alpha \in \overline{R} : Lab(\alpha) = UNDEC$;
10 $Lab' \leftarrow Lab$;
11 $Lab'(\alpha) \leftarrow IN$;
12 $Lab'(src(\alpha)) \leftarrow IN$;
13 **if** $trg(\alpha) \in A$ **then**
14 $Lab'(trg(\alpha)) \leftarrow OUT$;
15 **foreach** $\beta \in \overline{R} : src(\beta) = trg(\alpha)$ **do**
16 $Lab'(\beta) \leftarrow OUT$;
17 **else**
18 $Lab'(trg(\alpha)) \leftarrow OUT$;
19 **foreach** $\beta \in \overline{R} : Lab'(\beta) \in \{UNDEC, IGNORED\} \wedge trg(\beta) \in \{\alpha, src(\alpha)\}$ **do**
20 $Lab'(\beta) \leftarrow MUST\_OUT$;
21 **if** $\nexists\, \lambda \in \overline{R} : Lab'(\lambda) = UNDEC \wedge trg(\lambda) \in \{\beta, src(\beta)\}$ **then**
22 **if** $\exists \lambda \in \overline{R} : Lab(\lambda) = UNDEC \wedge trg(\lambda) \in \{\alpha, src(\alpha)\}$ **then**
23 $Lab(\alpha) \leftarrow IGNORED$;
24 **goto** line 8;
25 **else**
26 return;
27 **call** find-stable-extensions(Lab');
28 **if** $\exists \beta \in \overline{R} : Lab(\beta) = UNDEC \wedge trg(\beta) \in \{\alpha, src(\alpha)\}$ **then**
29 $Lab(\alpha) \leftarrow IGNORED$;
30 **else**
31 **if** $\nexists \beta \in \overline{R} : Lab(\beta) = IGNORED \wedge trg(\beta) \in \{\alpha, src(\alpha)\}$ **then**
32 $Lab \leftarrow Lab'$;
33 **else**
34 return;
35 **if** $\nexists \beta \in \overline{R} : Lab(\beta) \in \{MUST\_OUT, IGNORED\}$ **then**
36 **foreach** $x \in A : Lab(x) = UNDEC \wedge \forall \beta \in \overline{R} : trg(\beta) = x$ $(Lab(\beta) = OUT)$ **do**
37 $Lab(x) \leftarrow IN$;
38 $\overline{E}_{stable} \leftarrow \overline{E}_{stable} \cup \{x \in (A \cup \overline{R}) \mid Lab(x) = IN\}$;
39 **end procedure**

Table 6.2: The average elapsed time of algorithm 29 versus algorithm 14, instances of argument system with recursive attacks were generated randomly with $|A| = 35$ and by setting attacks with a probability p.

p	0.01	0.02	0.03	0.04	0.05
Average of $\|\overline{R}\|$	8.82	23.75	50.34	96.63	175.11
Algorithm 29	0.00	0.50	8.70	4,056.30	259,912.20
Algorithm 14	0.50	1.80	16.30	16,046.60	354,590.50

6.3 Enumerating Complete Extensions

Algorithm 30 decides all complete extensions. Indeed, algorithm 30 is a modification of algorithm 28, which decides preferred extensions. In algorithm 28 we find a preferred extension, say $pref$, if and only if for each $x \in A \cup \overline{R}$, x is not UNDEC nor MUST_OUT and $pref$ is not a subset of a previously decided preferred extension. However, in algorithm 30 we encounter a complete extension if and only if

$\nexists \beta \in \overline{R} : Lab(\beta) = MUST\_OUT$ *and*
$\nexists \lambda \in \overline{R} : Lab(\lambda) \in \{IGNORED, UNDEC\}$ *with* $\forall \alpha \in \overline{R} : trg(\alpha) \in \{\lambda, src(\lambda)\}\ Lab(\alpha) = OUT$.

Table 6.3 displays the performance of algorithm 30 versus algorithm 15.

6.4 Enumerating Stage Extensions

Algorithm 31 decides all stage extensions. In particular, algorithm 31 decides conflict free candidate subsets of $A \cup \overline{R}$ (see lines 15-29) by using the same five labels that are used in algorithm 28, which enumerates preferred extensions. Recall that algorithm 28 actually finds admissible sets rather than conflict free sets. Thus, algorithm 28 decides that elements in $A \cup \overline{R}$, which are labeled IN, make up an admissible set if and only if for each $x \in A \cup \overline{R}$, x is not UNDEC nor MUST_OUT. Algorithm 31 decides that elements in $A \cup \overline{R}$, which are labeled IN, make up a candidate conflict free set if and only if for each $x \in \overline{R}$ x is not UNDEC, see lines 15 & 32. Then, for every conflict free set S determined algorithm 31 also records $S' \equiv \{x \in A \cup \overline{R} \mid x \text{ is OUT}\}$. After accumulating all candidate $S \cup S'$, algorithm 31 decides that S is a stage extension if and only if $S \cup S'$ is maximal, see lines 8-12. As we stated in chapter 4, heuristics and pruning strategies used in semantics that are based on admissible sets will not be applicable to stage semantics, which are based on conflict free sets. Therefore, as a pruning strategy we skip ignoring an attack y if and only if

$\forall z \in \overline{R} : trg(z) \in \{y, src(y)\} \vee trg(y) \in \{z, src(z)\}$, z *is OUT or MUST_OUT or IGNORED*.

On selecting the next UNDEC attack to be labeled IN, we apply the following rule:

Algorithm 30: Enumerating complete extensions of an argument system with recursive attacks $(A, \overline{R})$.

1 $N(x) \equiv \begin{cases} 0 & \text{if } x \in \overline{R}; \\ |\{\beta \in \overline{R} \mid src(\beta) = x\}| & \text{if } x \in A; \end{cases}$
2 $Lab : (A \cup \overline{R}) \rightarrow \{IN, OUT, MUST\_OUT, IGNORED, UNDEC\}$; $Lab \leftarrow \phi$;
3 **foreach** $x \in A \cup \overline{R}$ **do** $Lab \leftarrow Lab \cup \{(x, UNDEC)\}$;
4 $\overline{E}_{complete} \subseteq 2^{A \cup \overline{R}}$; $\overline{E}_{complete} \leftarrow \phi$;
5 **call** find-complete-extensions(Lab);
6 **report** $\overline{E}_{complete}$ is the set of all complete extensions;

7 **procedure** find-complete-extensions(Lab)
8 **if** $\nexists \beta \in \overline{R} : Lab(\beta) = MUST\_OUT$ *and* $\nexists \lambda \in \overline{R} : Lab(\lambda) \in \{IGNORED, UNDEC\}$ *with* $\forall \alpha \in \overline{R} : trg(\alpha) \in \{\lambda, src(\lambda)\}$ $(Lab(\alpha) = OUT)$ **then**
9 **foreach** $x \in A : Lab(x) = UNDEC$ *and* $\forall \beta \in \overline{R} : trg(\beta) = x$ $(Lab(\beta) = OUT)$ **do**
10 $Lab(x) \leftarrow IN$;
11 $\overline{E}_{complete} \leftarrow \overline{E}_{complete} \cup \{x \in (A \cup \overline{R}) \mid Lab(x) = IN)\}$;
12 **while** $\exists \alpha \in \overline{R} : Lab(\alpha) = UNDEC$ **do**
13 select $\alpha \in \overline{R} : Lab(\alpha) = UNDEC$ and $\forall \beta \in \overline{R} : trg(\beta) \in \{\alpha, src(\alpha)\}$ $(Lab(\beta) \in \{OUT, MUST\_OUT\})$, else pick $\alpha \in \overline{R} : Lab(\alpha) = UNDEC$ and $Lab(trg(\alpha)) \neq OUT$ and $\forall \beta \in \overline{R} : Lab(\beta) = UNDEC$ $(N(trg(\alpha)) \geq N(trg(\beta)))$, otherwise select any $\alpha \in \overline{R} : Lab(\alpha) = UNDEC$;
14 $Lab' \leftarrow Lab$;
15 $Lab'(\alpha) \leftarrow IN$;
16 $Lab'(src(\alpha)) \leftarrow IN$;
17 **if** $trg(\alpha) \in A$ **then**
18 $Lab'(trg(\alpha)) \leftarrow OUT$;
19 **foreach** $\beta \in \overline{R} : src(\beta) = trg(\alpha)$ **do**
20 $Lab'(\beta) \leftarrow OUT$;
21 **else**
22 $Lab'(trg(\alpha)) \leftarrow OUT$;
23 **foreach** $\beta \in \overline{R} : Lab'(\beta) \in \{UNDEC, IGNORED\} \wedge trg(\beta) \in \{\alpha, src(\alpha)\}$ **do**
24 $Lab'(\beta) \leftarrow MUST\_OUT$;
25 **if** $\nexists\ \lambda \in \overline{R} : Lab'(\lambda) = UNDEC \wedge trg(\lambda) \in \{\beta, src(\beta)\}$ **then**
26 $Lab(\alpha) \leftarrow IGNORED$;
27 **goto** line 12;
28 **call** find-complete-extensions(Lab');
29 **if** $\exists \beta \in \overline{R} : Lab(\beta) \in \{UNDEC, IGNORED\} \wedge trg(\beta) \in \{\alpha, src(\alpha)\}$ **then**
30 $Lab(\alpha) \leftarrow IGNORED$;
31 **else**
32 $Lab \leftarrow Lab'$;
33 **end procedure**

Table 6.3: The average elapsed time of algorithm 30 versus algorithm 15, instances of argument system with recursive attacks were generated randomly with $|A| = 30$ and by setting attacks with a probability p.

p	0.01	0.02	0.03	0.04	0.05
Average of $\|\overline{R}\|$	5.59	15.46	31.03	58.07	98.26
Algorithm 30	0.00	0.40	1.40	844.50	361,490.20
Algorithm 15	0.00	0.00	1.70	1,105.50	384,902.10

1. select an UNDEC attack y s.t. for each $z \in \overline{R} : trg(z) \in \{y, src(y)\} \vee trg(y) \in \{z, src(z)\}$, z is OUT or MUST_OUT or IGNORED.

2. otherwise select an UNDEC attack y such that the number of attacks that are attacking y (or $src(y)$) alongside the attacks (or their source arguments) that are attacked by y is maximal.

The aim of the first part of this rule is to maximize the gain of the pruning strategy that skips ignoring y, which is the attack picked up according to the first part of the rule, based on the property that if a conflict free set, say S, will be decided while such y is ignored then $S \cup \{y\}$ is conflict free as well, and hence, there is no need to ignore y in the first place. Consequently, the earlier we label such y IN, the bigger part of the search space that will be pruned. Before coming to the second part of the heuristic rule, recall that the aim of heuristics in our algorithms is to accelerate reaching a goal state, which is a conflict free set in the case of algorithm 31. Note that such a goal state is reached if and only if for each $x \in \overline{R}$, x is not UNDEC. Thus, by applying the second part of the heuristic rule we maximize the number of OUT/MUST_OUT attacks and minimize the number of UNDEC attacks. Table 6.4 demonstrates the performance of algorithm 31 versus algorithm 16.

6.5 Enumerating Semi Stable Extensions

Algorithm 32 decides all semi stable extensions. In particular, algorithm 32 decides admissible sets (see lines 14-31) in the same way algorithm 28 does in enumerating preferred extensions. However, for every decided admissible set S algorithm 32 also determines $S' \equiv \{x \in A \cup \overline{R} \mid x \text{ is OUT}\}$. After accumulating all candidate $S \cup S'$, algorithm 32 decides that an admissible set S is a semi stable extension if and only if $S \cup S'$ is maximal, see lines 7-11. Table 6.5 shows the efficiency of algorithm 32 in comparison with algorithm 17.

Algorithm 31: Enumerating stage extensions of an argument system with recursive attacks $(A, \overline{R})$.

$N(x) \equiv \begin{cases} 0 & \text{if } x \in \overline{R}; \\ |\{\beta \in \overline{R} \mid src(\beta) = x\}| & \text{if } x \in A; \end{cases}$

Given $\beta \in \overline{R}$, then $O(\beta) \equiv |\{\alpha \in \overline{R} \mid trg(\alpha) \in \{\beta, src(\beta)\}\}|$;

$Lab : (A \cup \overline{R}) \rightarrow \{IN, OUT, MUST\_OUT, IGNORED, UNDEC\}$; $Lab \leftarrow \phi$;

foreach $x \in A \cup \overline{R}$ **do** $Lab \leftarrow Lab \cup \{(x, UNDEC)\}$;

$\overline{E}_{stage} : ((A \cup \overline{R}) \rightarrow \{IN, OUT, MUST\_OUT, IGNORED, UNDEC\}) \times \mathbb{Z}$;

$\overline{E}_{stage} \leftarrow \phi$;

call find-conflict-free-sets(Lab);

`/* The next loop is to collect conflict free sets, those which have a maximal range */`

foreach $(Lab_1, i) \in \overline{E}_{stage}$ **do**

  foreach $(Lab_2, j) \in \overline{E}_{stage} : j \neq i$ **do**

    if $(|\{x : Lab_1(x) \in \{IN, OUT\}\}| \neq |\{z : Lab_2(z) \in \{IN, OUT\}\}| \vee Lab_1 = Lab_2)$ *and* $\forall y \in A \cup \overline{R} : Lab_1(y) \in \{IN, OUT\}$ $(Lab_2(y) \in \{IN, OUT\})$ **then**

      $\overline{E}_{stage} \leftarrow \overline{E}_{stage} \setminus \{(Lab_1, i)\}$;

      continue to next iteration from line 8;

foreach $(Lab_1, i) \in \overline{E}_{stage}$ **report** $\{x : Lab_1(x) = IN\}$ as a stage extension ;

procedure find-conflict-free-sets(Lab)

while $\exists \alpha \in \overline{R} : Lab(\alpha) = UNDEC$ **do**

  select $\alpha \in \overline{R} : Lab(\alpha) = UNDEC$ and $\forall \beta \in \overline{R} : trg(\beta) \in \{\alpha, src(\alpha)\} \vee trg(\alpha) \in \{\beta, src(\beta)\}$ $(Lab(\beta) \in \{OUT, MUST\_OUT, IGNORED\})$, else pick $\alpha \in \overline{R} : Lab(\alpha) = UNDEC$ and $Lab(trg(\alpha)) \neq OUT$ and $\forall \beta \in \overline{R} : Lab(\beta) = UNDEC$ $((N(trg(\alpha)) + O(\alpha)) \geq (N(trg(\beta)) + O(\beta)))$, otherwise select any $\alpha \in \overline{R} : Lab(\alpha) = UNDEC$;

  $Lab' \leftarrow Lab$; $Lab'(\alpha) \leftarrow IN$; $Lab'(src(\alpha)) \leftarrow IN$;

  if $trg(\alpha) \in A$ **then**

    $Lab'(trg(\alpha)) \leftarrow OUT$;

    foreach $\beta \in \overline{R} : src(\beta) = trg(\alpha)$ **do** $Lab'(\beta) \leftarrow OUT$;

  else

    $Lab'(trg(\alpha)) \leftarrow OUT$;

  foreach $\beta \in \overline{R} : Lab'(\beta) \in \{UNDEC, IGNORED\} \wedge trg(\beta) \in \{\alpha, src(\alpha)\}$ **do**

    $Lab'(\beta) \leftarrow MUST\_OUT$;

  call find-conflict-free-sets(Lab');

  if $\exists \beta \in \overline{R} : Lab(\beta) = UNDEC$ *and* $(trg(\beta) \in \{\alpha, src(\alpha)\} \vee trg(\alpha) \in \{\beta, src(\beta)\})$ **then**

    $Lab(\alpha) \leftarrow IGNORED$;

  else

    $Lab \leftarrow Lab'$;

foreach $x \in A : Lab(x) = UNDEC$ *and* $\forall \beta \in \overline{R} : trg(\beta) = x$ $(Lab(\beta) = OUT)$ **do**

  $Lab(x) \leftarrow IN$;

$\overline{E}_{stage} \leftarrow \overline{E}_{stage} \cup \{(Lab, |\overline{E}_{stage}| + 1)\}$;

end procedure

Algorithm 32: Enumerating semi stable extensions of an argument system with recursive attacks $(A, \overline{R})$.

1 $N(x) \equiv \begin{cases} 0 & \text{if } x \in \overline{R}; \\ |\{\beta \in \overline{R} \mid src(\beta) = x\}| & \text{if } x \in A; \end{cases}$

2 $Lab : (A \cup \overline{R}) \rightarrow \{IN, OUT, MUST\_OUT, IGNORED, UNDEC\}$; $Lab \leftarrow \phi$;

3 **foreach** $x \in A \cup \overline{R}$ **do** $Lab \leftarrow Lab \cup \{(x, UNDEC)\}$;

4 $\overline{E}_{semistable} : ((A \cup \overline{R}) \rightarrow \{IN, OUT, MUST\_OUT, IGNORED, UNDEC\}) \times \mathbb{Z}$;

5 $\overline{E}_{semistable} \leftarrow \phi$;

6 **call** find-admissible-sets(Lab);

/* The next loop is to pick up admissible sets, those which have a maximal range */

7 **foreach** $(Lab_1, i) \in \overline{E}_{semistable}$ **do**

8 **foreach** $(Lab_2, j) \in \overline{E}_{semistable} : j \neq i$ **do**

9 **if** $(|\{x : Lab_1(x) \in \{IN, OUT\}\}| \neq |\{z : Lab_2(z) \in \{IN, OUT\}\}| \vee Lab_1 = Lab_2)$ *and* $\forall y \in A \cup \overline{R} : Lab_1(y) \in \{IN, OUT\}$ $(Lab_2(y) \in \{IN, OUT\})$ **then**

10 $\overline{E}_{semistable} \leftarrow \overline{E}_{semistable} \setminus \{(Lab_1, i)\}$;

11 continue to next iteration from line 7;

12 **foreach** $(Lab_1, i) \in \overline{E}_{semistable}$ **report** $\{x : Lab_1(x) = IN\}$ as a semi stable extension ;

13 **procedure** find-admissible-sets(Lab)

14 **while** $\exists \alpha \in \overline{R} : Lab(\alpha) = UNDEC$ **do**

15 select $\alpha \in \overline{R} : Lab(\alpha) = UNDEC$ and $\forall \beta \in \overline{R} : trg(\beta) \in \{\alpha, src(\alpha)\}$ $(Lab(\beta) \in \{OUT, MUST\_OUT\})$, else pick $\alpha \in \overline{R} : Lab(\alpha) = UNDEC$ and $Lab(trg(\alpha)) \neq OUT$ and $\forall \beta \in \overline{R} : Lab(\beta) = UNDEC$ $(N(trg(\alpha)) \geq N(trg(\beta)))$, otherwise select any $\alpha \in \overline{R} : Lab(\alpha) = UNDEC$;

16 $Lab' \leftarrow Lab$; $Lab'(\alpha) \leftarrow IN$; $Lab'(src(\alpha)) \leftarrow IN$;

17 **if** $trg(\alpha) \in A$ **then**

18 $Lab'(trg(\alpha)) \leftarrow OUT$;

19 **foreach** $\beta \in \overline{R} : src(\beta) = trg(\alpha)$ **do** $Lab'(\beta) \leftarrow OUT$;

20 **else**

21 $Lab'(trg(\alpha)) \leftarrow OUT$;

22 **foreach** $\beta \in \overline{R} : Lab(\beta) \in \{UNDEC, IGNORED\} \wedge trg(\beta) \in \{\alpha, src(\alpha)\}$ **do**

23 $Lab'(\beta) \leftarrow MUST\_OUT$;

24 **if** $\nexists \lambda \in \overline{R} : Lab'(\lambda) = UNDEC \wedge trg(\lambda) \in \{\beta, src(\beta)\}$ **then**

25 $Lab(\alpha) \leftarrow IGNORED$;

26 **goto** line 14;

27 **call** find-admissible-sets(Lab');

28 **if** $\exists \beta \in \overline{R} : Lab(\beta) \in \{UNDEC, IGNORED\} \wedge trg(\beta) \in \{\alpha, src(\alpha)\}$ **then**

29 $Lab(\alpha) \leftarrow IGNORED$;

30 **else**

31 $Lab \leftarrow Lab'$;

32 **if** $\nexists \beta \in \overline{R} : Lab(\beta) = MUST\_OUT$ **then**

33 **foreach** $x \in A : Lab(x) = UNDEC$ *and* $\forall \beta \in \overline{R} : trg(\beta) = x$ $(Lab(\beta) = OUT)$ **do**

34 $Lab(x) \leftarrow IN$;

35 $\overline{E}_{semistable} \leftarrow \overline{E}_{semistable} \cup \{(Lab, |\overline{E}_{semistable}| + 1)\}$;

36 **end procedure**

Table 6.4: The average elapsed time of algorithm 31 versus algorithm 16, instances of argument system with recursive attacks were generated randomly with $|A| = 15$ and by setting attacks with a probability p.

p	0.01	0.02	0.03	0.04	0.05
Average of $\|\overline{R}\|$	1.13	2.80	4.52	7.44	10.54
Algorithm 31	0.20	0.30	0.70	22.90	583.90
Algorithm 16	0.10	1.30	3.60	153.00	1,832.80

Table 6.5: The average elapsed time of algorithm 32 versus algorithm 17, instances of argument system with recursive attacks were generated randomly with $|A| = 25$ and by setting attacks with a probability p.

p	0.01	0.02	0.03	0.04	0.05	0.06
Average of $\|\overline{R}\|$	7.95	18.77	31.89	47.81	66.59	86.77
Algorithm 32	10.40	9.00	10.00	11.20	29.40	395.10
Algorithm 17	10.60	9.80	10.60	11.50	49.80	611.70

6.6 Deciding the Grounded Extension

Algorithm 33 decides the grounded extension. As is the case in all algorithms in this chapter, algorithm 33 can be seen as a generalization of algorithm 19. Table 6.6 shows the efficiency of algorithm 33 in comparison with algorithm 19.

6.7 Deciding the Ideal Extension

Algorithm 34 decides the ideal extension. In particular, algorithm 34 decides admissible sets (lines 11-31) in the same way algorithm 28 does in deciding preferred extensions. However, in enumerating admissible sets algorithm 34 also determines S that is described as

$$S \equiv \{x \in A \cup \overline{R} \mid \textit{there is } y \textit{ in an admissible set s.t. } y \in \overline{R} \textit{ and } trg(y) \in \{x, src(x)\}\}$$

After enumerating a set of candidate admissible sets and having S determined, algorithm 34 decides that an admissible set T is the ideal extension if and only if $T \cap S = \phi$, see lines 7-9. Recall that all algorithms, including algorithm 34, decide admissible sets in an ascending order, i.e. from the largest set to the smallest one. Table 6.7 reports the efficiency of algorithm 34 in comparison with algorithm 18.

Algorithm 33: Deciding the grounded extension of an argument system with recursive attacks $(A, \overline{R})$.

```
1  Lab : (A ∪ R̄) → {IN, OUT, UNDEC};  Lab ← φ;
2  foreach w ∈ A ∪ R̄ do Lab ← Lab ∪ {(w, UNDEC)};
3  while ∃x ∈ R̄ : Lab(x) = UNDEC ∧ ∀y ∈ R̄ : trg(y) ∈ {x, src(x)} (Lab(y) = OUT) do
4      foreach x ∈ R̄ : Lab(x) = UNDEC ∧ ∀y ∈ R̄ : trg(y) ∈ {x, src(x)} (Lab(y) = OUT) do
5          Lab(x) ← IN;
6          Lab(src(x)) ← IN;
7          Lab(trg(x)) ← OUT;
8          if trg(x) ∈ A then
9              foreach z ∈ R̄ : trg(x) = src(z) do
10                 Lab(z) ← OUT;
11 foreach  x ∈ A : Lab(x) = UNDEC ∧ ∀β ∈ R̄ : trg(β) = x (Lab(β) = OUT) do
12     Lab(x) ← IN;
13 report the grounded extension is {w ∈ A ∪ R̄ | Lab(w) = IN};
```

Table 6.6: The average elapsed time of algorithm 33 versus algorithm 19, instances of argument system with recursive attacks were generated randomly with $|A| = 100$ and by setting attacks with a probability p.

p	0.01	0.02	0.03	0.04	0.05
Average of $\lvert\overline{R}\rvert$	132.89	738.87	3,071.67	11,546.59	41,471.95
Algorithm 33	5.80	10.00	25.20	63.80	207.10
Algorithm 19	9.50	18.90	46.20	144.00	521.00

Algorithm 34: Deciding the ideal extension of an argument system with recursive attacks $(A, \overline{R})$.

1 let $N(x) \equiv \begin{cases} 0 & \text{if } x \in \overline{R}; \\ |\{\beta \in \overline{R} \mid src(\beta) = x\}| & \text{if } x \in A; \end{cases}$
2 $Lab : (A \cup \overline{R}) \to \{IN, OUT, MUST\_OUT, IGNORED, UNDEC\}$; $Lab \leftarrow \phi$;
3 **foreach** $x \in A \cup \overline{R}$ **do** $Lab \leftarrow Lab \cup \{(x, UNDEC)\}$;
4 $E_{ideal} : \mathbb{Z} \to 2^{A \cup \overline{R}}$; $E_{ideal} \leftarrow \phi$;
5 $S \leftarrow \phi$;
/* S will hold the arguments/attacks that certainly are not in the ideal extension */
6 **call** find-admissible-sets(Lab);
7 **foreach** i : 1 *to* $|E_{idea}|$ **do**
8 **if** $\forall x \in E_{ideal}(i)$ $(x \notin S)$ **then**
9 **report** $E_{ideal}(i)$ is the ideal extension; **exit**;

10 **procedure** find-admissible-sets(Lab)
11 **while** $\exists \alpha \in \overline{R} : Lab(\alpha) = UNDEC$ **do**
12 select $\alpha \in \overline{R} : Lab(\alpha) = UNDEC$ and
$\forall \beta \in \overline{R} : trg(\beta) \in \{\alpha, src(\alpha)\}$ $(Lab(\beta) \in \{OUT, MUST\_OUT\})$, else pick
$\alpha \in \overline{R} : Lab(\alpha) = UNDEC$ and $Lab(trg(\alpha)) \neq OUT$ and $\forall \beta \in \overline{R} : Lab(\beta) = UNDEC$
$(N(trg(\alpha)) \geq N(trg(\beta)))$, otherwise select any $\alpha \in \overline{R} : Lab(\alpha) = UNDEC$;
13 $Lab' \leftarrow Lab$;
14 $Lab'(\alpha) \leftarrow IN$;
15 $Lab'(src(\alpha)) \leftarrow IN$;
16 **if** $trg(\alpha) \in A$ **then**
17 $Lab'(trg(\alpha)) \leftarrow OUT$;
18 **foreach** $\beta \in \overline{R} : src(\beta) = trg(\alpha)$ **do**
19 $Lab'(\beta) \leftarrow OUT$;
20 **else**
21 $Lab'(trg(\alpha)) \leftarrow OUT$;
22 **foreach** $\beta \in \overline{R} : Lab'(\beta) \in \{UNDEC, IGNORED\} \wedge trg(\beta) \in \{\alpha, src(\alpha)\}$ **do**
23 $Lab'(\beta) \leftarrow MUST\_OUT$;
24 **if** $\nexists\, \lambda \in \overline{R} : Lab'(\lambda) = UNDEC \wedge trg(\lambda) \in \{\beta, src(\beta)\}$ **then**
25 $Lab(\alpha) \leftarrow IGNORED$;
26 **goto** line 11;
27 **call** find-admissible-sets(Lab');
28 **if** $\exists \beta \in \overline{R} : Lab(\beta) \in \{UNDEC, IGNORED\}$ *and* $trg(\beta) \in \{\alpha, src(\alpha)\}$ **then**
29 $Lab(\alpha) \leftarrow IGNORED$;
30 **else**
31 $Lab \leftarrow Lab'$;
32 **if** $\nexists \beta \in \overline{R} : Lab(\beta) = MUST\_OUT$ **then**
33 **foreach** $x \in A : Lab(x) = UNDEC$ *and* $\forall \beta \in \overline{R} : trg(\beta) = x$ $(Lab(\beta) = OUT)$ **do**
34 $Lab(x) \leftarrow IN$;
35 $S \leftarrow S \cup \{x \in (A \cup \overline{R}) \mid Lab(x) = OUT\}$;
36 $E_{ideal} \leftarrow E_{ideal} \cup \{(|E_{ideal}| + 1, \{z \in (A \cup \overline{R}) \mid Lab(z) = IN\})\}$;
37 **end procedure**

Table 6.7: The average elapsed time of algorithm 34 versus algorithm 18, instances of argument system with recursive attacks were generated randomly with $|A| = 35$ and by setting attacks with a probability p.

p	0.01	0.02	0.03	0.04	0.05
Average of $\|\overline{R}\|$	8.65	23.31	51.24	99.97	174.24
Algorithm 34	0.10	0.10	5,064.20	166,875.70	817,050.40
Algorithm 18	0.50	0.00	9,347.20	175,207.40	709,018.50

6.8 Summary

We presented algorithms for enumerating extensions under several semantics in argument systems with recursive attacks. Thus, we offered a unified approach to enumerating extensions for both standard argument systems and argument systems with recursive attacks. This follows from the fact that Dung argument systems are a special case of argument systems with recursive attacks [7]. By doing so, we showed how labeling attacks alongside arguments can be used as a basis for enumerating extensions in argument systems, especially those formalisms that allow attacks on attacks (e.g. [76, 57, 7]). In fact, extensions of an argument system expressed in such formalisms can be computed by firstly translating the argument system back into a standard Dung argument system. Nonetheless, we demonstrated how labeling can be applied directly on the native form of such formalisms to enumerate extensions without compromising efficiency. Turning to related works, [75] defined labeling-based semantics for Modgil's argument system introduced in [76] while [98] described argumentation semantics in terms of attacks rather than arguments in the context of Dung argument systems. [57] set semantics for the extended argument systems of [11] that also allow attacks on attacks.

Chapter 7

Conclusion

In this chapter we close the thesis. Section 7.1 summarizes the contributions by reviewing the algorithms we engineered. In section 7.2 we discuss further related works while in section 7.3 we present directions for future developments.

7.1 Thesis Review

In the present work we stressed that experimental analysis is essential in engineering algorithms in the context of argument systems. We motivated experimental algorithms by presenting two case studies. The first case study was to compare the performance of different algorithmic methods for computing an argument's acceptance in the setting of Modgil's argument system [76], whereas the second case study was to analyze the behavior of an algorithm for deciding acceptance in value based argument systems proposed by Bench-Capon [13].

In addition we developed a novel concrete algorithm for preferred semantics. Existing labeling-based algorithms use three labels while our algorithm employs further labels to enhance efficiency. Essentially, our algorithm is characterized by applying powerful pruning mechanisms and cost-effective heuristic rules. Also, our algorithm simplifies ensuring the maximality, which is a core property of preferred extensions, via exploring candidate sets in descending order. After giving an analytical comparison between our algorithm and the existing algorithms of [35, 78], we reported experimental results indicating that our algorithm is superior with respect to the running time. Afterwards, we modified our algorithm to answer credulous/skeptical acceptance without explicitly enumerating all preferred extensions. Then, we introduced more comparisons with the existing algorithms of [28, 95, 97], which are dedicated to deciding credulous/skeptical acceptance. As further evidence we documented empirical results indicating that our algorithm for credulous/skeptical acceptance is more efficient than the algorithms of [28, 95, 97].

As it might be expected, our algorithm for preferred semantics can be altered to find extensions under other semantics. Therefore, we engineered precise algorithms for sta-

ble, semi stable, stage, complete, ideal semantics. As we have shown, the designed algorithms for admissibility-based semantics (i.e. preferred, semi stable, complete and ideal semantics) make use of the same pruning mechanisms and heuristics. For stable semantics we exploit a further pruning property besides the strategies used in admissibility-based semantics. For stage semantics, which is a conflict-freeness-based semantics, we illustrated that those pruning techniques and heuristics applied in admissibility-based semantics are not effective, and thus, we utilized slightly different pruning techniques and heuristics. To explore the efficiency of the developed algorithms we compared with ASPARTIX [56], which is an implemented system for solving decision problems under several argumentation semantics.

The objective/subjective acceptance problem in value based argument systems is computationally more challenging than credulous/skeptical acceptance in standard argument systems. This is because, to answer objective/subjective acceptance one has to evaluate preferred extensions for every total value order. However, we established that it might be the case that there are some redundant total value orders that can be safely ignored. Thus, in this regard we developed labeling-based algorithms for deciding objective/subjective acceptance by enumerating preferred extensions w.r.t. only "critical" total value orders. Similarly, we have set an efficient approach to encoding total orders over the set of social values, V, such that the space complexity is upper bounded to the squared number of values (i.e. $|V|^2$) rather than to the number of all total value orders (i.e. $|V!|$), which is the case if a naive approach is adopted.

The concept of allowing attacks on attacks, i.e. informally allowing a given attack to attack an attack, has been a central motivation for developing a number of argumentation formalisms (see e.g. [76, 57, 7]). In fact, the algorithms designed for standard argument systems are not applicable directly to such formalisms. Therefore, we designed a generalization to our earlier algorithms such that attacks along with arguments are subject to labeling rather than labeling arguments only. Subsequently, under a number of prevalent argumentation semantics we ended up with algorithms that operate on standard argument systems as well as on argument systems with recursive attacks [7], which is an instance of formalisms that allow attacks on attacks. We verified experimentally that enumerating extensions by labeling attacks besides arguments is as efficient as listing extensions by labeling arguments alone.

7.2 Further Related Works

We stress that we compare our algorithms with the existing algorithms of [35, 78, 28, 97, 95] analytically and empirically. Also, we show the efficiency of our algorithms against the implemented systems of dynPARTIX and ASPARTIX experimentally. In each chapter we reviewed closely related works respectively in the summary section.

However, in this section we outline some other available works which tackle problems in the field of computational argumentation other than the problems addressed by our algorithms. We aim by this section to give another perspective to the topic of this research. Thus, we do not present deep analysis for the literature listed in the following overview.

To start with, in the setting of so-called "dynamic abstract argumentation frameworks" (DAFs for short) [93], [92] defined heuristic-based pruning rule that might help in improving the efficiency of deciding the acceptance of arguments. A main idea in DAFs is that the universal set of arguments is divided into two parts: active and inactive. To decide the acceptance of a given argument with respect to the active arguments, a pruning technique is applied based on a heuristic function that reflects the strength of arguments in the universal set. [92] stated that computing arguments' strength is expensive given the nature of their heuristic function. However, [92] argued after conducting experiments that their pruning strategy was successful in enhancing the performance under the assumption that active/inactive arguments are dynamic while the universal set is static. Thus, the one-off cost of computing the heuristic value for all arguments in the universal set will be paid off over time according to [92].

[61] introduced a formalism, so-called "DeLP", that combines logic programming with defeasible argumentation, and so, [61] presented a dialectical proof procedure for deciding the warrant (i.e. acceptance) of a given argument.

In the context of Dung's argument system the work of [94] discussed the impact on the status of arguments in a "dialectical tree" if some argument is dropped from the tree while [30] specified a model for so-called the "exhaustive dialectical tree" of an argument. the later model is somewhat similar to the base concept of [95] which is an essential, but not sufficient, condition for skeptical acceptance.

The works of [17] and [55] are concerned with building algorithms for finding logic-based arguments/counterarguments from a knowledgebase.

[59, 38] presented dialectical proof procedures for a specific instantiation of Dung's argument system: so-called "assumption based argumentation frameworks". In fact these dialectical procedures use similar notions of the procedures of [95] which we have discussed in chapter 3.

The work of [60] proposed an algorithm that translates value based argument systems to neural networks. The formalism of [65] extended value based argument systems such that arguments promote multiple social values rather than one value, and then, [65] defined an algorithm for computing the grounded extension in the new model.

The algorithm of [49] finds extensions under so-called "cf2" argumentation semantics. [66] described proof theories for deciding credulous acceptance under preferred semantics. Finally, [9] suggested a distributed approach to argumentation.

7.3 Future Developments

This work can be taken further in several directions:

- We plan to expand our work to cover other Dung's style argumentation semantics such as "resolution-based semantics" [8], "cf2" semantics [10], "stage2" semantics [50], "prudent" semantics [32] and "eager" semantics [26].
- We intend to engineer algorithms for other argumentation systems that extend Dung's model such as argument systems with "varied strength attacks" [72, 46], "bipolar" argument systems [29], "weighted" argument systems [44], "uniform argumentation frameworks" [4], and argument systems with "higher order attacks" [57].
- We aim to set up concrete algorithms for other decision problems in the context of argument systems such as "argument aggregation" (see e.g. [31, 45]) and "dynamics" of argument systems (see e.g. [71, 93]).
- We plan to build algorithms for other argumentation formalisms such as the systems of [22] so-called "assumption based argumentation frameworks", the frame of [88] so-called "structured argumentation frameworks", and logic-based argumentation systems [18].
- In all the above perspectives the main research question is around constructing formal search algorithms that feature essentially three efficiency elements:
  1. a powerful mechanism through which the underlying system transitions from a state (i.e. node of the search tree) to another state.
  2. a pruning strategy by which branches of the search tree are bypassed.
  3. heuristics that might guide the search process to the shortest path leading to the concerned goal state.
- The dialectical nature of argumentation explains why argument systems provide a backbone for the study of agent dialogs, see [73] for a general review. Hence, we see that the notion of transitions, heuristics and pruning tactics presented in this work are all potentially applicable in improving the efficiency of agent interactions in different scenarios. Nonetheless, this is to be seen by doing further concrete research.
- In this work we only explored three possible heuristic rules for selecting the next argument to be included in a candidate extension. Our intention was to keep heuristics as simple as possible to ensure cost-effectiveness. However, it would be worth investigating the effectiveness of other applicable selection rules that might be considered quite sophisticated.

Appendix A

Data Tables for Charts in Chapter 2

We provide tables A.1- A.5 that are traced by figures presented in section 2.1.2. Recall that $|(A,R,D)|$ refers to the number of instances of Modgil's argument system. Also, tables A.6 & A.7 are the data for the figures in section 2.2.2.

Table A.1: Data traced by figures 2.6 and 2.7.

$\|R\|$	$\|(A,R,D)\|$	Credulous Acceptance	Skeptical Acceptance
12	73	1606.29	16675.35
24	102	676.11	5560.71
36	87	367.24	2831.97
48	98	210.32	1992.07
60	87	132.65	1380.45
72	106	76.43	1030.04
84	75	58.53	713.38
96	93	51.04	586.36
108	111	26.89	363.77
120	82	7.01	164.71
132	86	43.14	80.13

Table A.2: Data traced by figures 2.8 and 2.9.

$\|D\|$	$\|(A,R,D)\|$	Credulous Acceptance	Skeptical Acceptance
12	98	76.13	894.17
24	82	192.73	1888.52
36	86	332.49	3295.78
48	93	530.73	4700.22
60	90	723.45	5961.42
72	101	1124.95	7825.10
84	84	1435.98	9758.77
96	82	2039.03	11627.13
108	87	2103.00	13596.98
120	91	2488.44	14595.18
132	106	3054.75	17240.23

Table A.3: Trend of α_{time} traced by figures 2.2 and 2.4.

$\|A\|$	$\|(A,R,D)\|$	Credulous Acceptance			Skeptical Acceptance		
		method 1	method 2	method 3	method 1	method 2	method 3
6	59	0.11	0.01	0.00	0.59	0.05	0.03
7	58	0.19	0.04	0.03	1.14	0.28	0.26
8	60	0.38	0.05	0.06	2.47	0.33	0.40
9	82	0.74	0.14	0.10	5.34	1.00	0.88
10	82	1.30	0.22	0.17	10.68	2.10	1.83
11	67	2.31	0.40	0.39	16.90	4.54	4.22
12	78	4.20	0.91	0.71	28.49	9.18	7.99
13	56	5.81	1.64	1.10	50.61	15.75	13.09
14	66	10.27	3.14	2.23	97.95	24.55	20.48
15	73	20.01	5.14	4.08	207.88	43.71	39.82
16	75	29.00	7.29	5.87	431.88	86.04	78.55
17	65	64.29	13.26	11.92	904.14	171.09	161.51
18	55	107.93	19.72	17.74	2089.15	355.98	336.53
19	61	260.89	45.58	42.06	4402.82	720.59	687.08
20	63	543.86	94.83	87.46	9378.41	1597.16	1523.00

Table A.4: Trend of α_d traced by figures 2.3 and 2.5.

$\|A\|$	$\|(A,R,D)\|$	Credulous Acceptance			Skeptical Acceptance		
		method 1	method 2	method 3	method 1	method 2	method 3
6	59	336.07	11.08	15.54	1498.64	40.41	71.07
7	58	673.63	16.21	28.60	3304.17	73.03	165.74
8	60	1217.52	27.48	59.78	6836.10	145.38	386.58
9	82	2274.21	45.06	113.70	14811.65	302.40	994.82
10	82	3501.13	62.54	182.02	28183.44	488.74	1793.34
11	67	6540.31	106.19	352.07	56080.57	902.18	3681.01
12	78	12296.06	168.41	672.05	109885.96	1648.50	7699.22
13	56	21324.02	282.00	1109.52	238857.05	3524.80	17659.18
14	66	47516.56	628.24	3151.11	489937.58	6611.92	37132.82
15	73	99504.17	1195.12	6275.61	1089272.55	13456.51	79450.23
16	75	134600.25	1720.37	9947.99	2146528.56	27514.96	182848.55
17	65	317136.59	3324.86	19759.54	4600649.11	52010.05	348581.34
18	55	502602.26	5110.63	33008.86	9820289.53	98532.38	731753.42
19	61	1252398.48	10388.59	72271.21	20347795.93	168775.64	1238087.48
20	63	2633976.96	24870.89	182065.24	43625689.46	429361.43	3438985.17

Table A.5: Trend of average of procedure invocations traced by figures 2.10 and 2.11.

$\|A\|$	$\|(A,R,D)\|$	Credulous Acceptance		Skeptical Acceptance	
		method 2	method 3	method 2	method 3
6	59	19.75	10.63	57.22	30.69
7	58	23.55	12.54	75.91	40.36
8	60	32.89	17.54	114.42	61.22
9	82	46.01	24.52	176.38	94.41
10	82	55.68	29.49	222.85	118.07
11	67	74.25	39.42	325.88	174.33
12	78	90.26	47.51	375.45	199.08
13	56	126.24	66.43	550.36	290.84
14	66	147.26	77.27	683.53	359.95
15	73	197.75	103.84	970.84	512.59
16	75	245.42	128.95	1307.55	689.51
17	65	336.99	176.93	1787.48	941.37
18	55	350.34	182.65	2020.67	1057.09
19	61	498.77	258.84	2753.92	1434.75
20	63	616.76	320.26	3624.71	1892.14

Table A.6: Data traced by figures 2.22 and 2.23.

$\|V\|$	$\|$*value based argument systems*$\|$	$\alpha_{value-order}$	α_{time}
2	340	1.94	1.32
3	350	2.56	2.37
4	354	3.14	3.10
5	356	3.66	4.08
6	340	4.14	5.94
7	355	4.43	8.75
8	394	4.87	11.23
9	431	5.55	13.91
10	466	7.19	18.73
11	557	10.97	24.57
12	551	15.41	30.53
13	589	33.64	40.18
14	646	52.66	61.59
15	724	83.22	82.46
16	778	160.87	134.18
17	816	287.60	201.12
18	769	430.10	331.25
19	567	759.19	499.55
20	370	1,186.06	754.96

Table A.7: Data traced by figures 2.24 and 2.25.

$\|attacks\ per\ argument\|$	$\alpha_{value-order}$	α_{time}
2	35.84	0.05
3	53.67	0.48
4	62.63	1.47
5	66.28	4.19
6	68.63	8.67
7	70.37	13.22
8	71.24	18.76
9	72.35	26.19
10	73.24	35.04
11	72.54	41.82
12	73.78	52.03
13	73.92	64.02
14	73.59	69.70
15	74.52	82.64
16	75.39	91.84
17	75.07	99.70
18	74.71	104.36
19	75.38	114.39
20	75.67	123.15

Bibliography

[1] L. Amgoud and C. Cayrol. A reasoning model based on the production of acceptable arguments. *Annals of Mathematics and Artificial Intelligence*, 34:197–215, 2002.

[2] L. Amgoud and C. Devred. Argumentation frameworks as constraint satisfaction problems. In S. Benferhat and J. Grant, editors, *Scalable Uncertainty Management - 5th International Conference, SUM 2011, Proceedings*, volume 6929 of *Lecture Notes in Computer Science*, pages 110–122. Springer, 2011.

[3] L. Amgoud and S. Parsons. An argumentation framework for merging conflicting knowledge bases. In S. Flesca, S. Greco, N. Leone, and G. Ianni, editors, *Logics in Artificial Intelligence, European Conference, JELIA 2002, Proceedings*, volume 2424 of *Lecture Notes in Computer Science*, pages 27–37. Springer, 2002.

[4] K. Atkinson, T.J.M. Bench-Capon, and P.E. Dunne. Uniform argumentation frameworks. In B. Verheij, S. Szeider, and S. Woltran, editors, *Computational Models of Argument - Proceedings of COMMA 2012*, volume 245 of *Frontiers in Artificial Intelligence and Applications*, pages 165–176. IOS Press, 2012.

[5] K. Atkinson, T.J.M. Bench-Capon, and S. Modgil. Argumentation for decision support. In S. Bressan, J. Küng, and R. Wagner, editors, *Database and Expert Systems Applications, 17th International Conference, DEXA 2006, Proceedings*, volume 4080 of *Lecture Notes in Computer Science*, pages 822–831. Springer, 2006.

[6] P. Baroni, M. Caminada, and M. Giacomin. An introduction to argumentation semantics. *The Knowledge Engineering Review*, 26(4):365–410, 2011.

[7] P. Baroni, F. Cerutti, M. Giacomin, and G. Guida. Argumentation framework with recursive attacks. *International Journal of Approximate Reasoning*, 52(1):19–37, 2011.

[8] P. Baroni, P.E. Dunne, and M. Giacomin. On the resolution-based family of abstract argumentation semantics and its grounded instance. *Artificial Intelligence*, 175(3-4):791 – 813, 2011.

[9] P. Baroni and M. Giacomin. Argumentation through a distributed self-stabilizing approach. *Journal of Experimental and Theoretical Artificial Intelligence*, 14(4):273–301, 2002.

[10] P. Baroni, M. Giacomin, and G. Guida. Scc-recursiveness: a general schema for argumentation semantics. *Artificial Intelligence*, 168(1-2):162 – 210, 2005.

[11] H. Barringer, D.M. Gabbay, and J. Woods. Temporal dynamics of support and attack networks: From argumentation to zoology. In D. Hutter and W. Stephan, editors, *Mechanizing Mathematical Reasoning*, volume 2605 of *Lecture Notes in Computer Science*, pages 59–98. Springer, 2005.

[12] R. Baumann, G. Brewka, and R. Wong. Splitting argumentation frameworks: An empirical evaluation. In S. Modgil, N. Oren, and F. Toni, editors, *Theory and Applications of Formal Argumentation - First International Workshop, TAFA 2011*, volume 7132 of *Lecture Notes in Computer Science*, pages 17–31. Springer, 2011.

[13] T.J.M. Bench-Capon. Persuasion in practical argument using value-based argumentation frameworks. *J. Log. Comput.*, 13(3):429–448, 2003.

[14] T.J.M. Bench-Capon, K. Atkinson, and A. Chorley. Persuasion and value in legal argument. *J. Log. Comput.*, 15(6):1075–1097, 2005.

[15] T.J.M. Bench-Capon and P.E. Dunne. Argumentation in artificial intelligence. *Artificial Intelligence*, 171:619–641, 2007.

[16] P. Besnard and S. Doutre. Checking the acceptability of a set of arguments. In J.P. Delgrande and T. Schaub, editors, *10th International Workshop on Non-Monotonic Reasoning (NMR 2004), Proceedings*, pages 59–64, 2004.

[17] P. Besnard and A. Hunter. Knowledgebase compilation for efficient logical argumentation. In P. Doherty, J. Mylopoulos, and C.A. Welty, editors, *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning*, pages 123–133. AAAI Press, 2006.

[18] P. Besnard and A. Hunter. *Elements of Argumentation*. MIT press, 2008.

[19] E. Black and K. Bentley. An empirical study of a deliberation dialogue system. In S. Modgil, N.Oren, and F. Toni, editors, *Theory and Applications of Formal Argumentation - First International Workshop, TAFA 2011*, volume 7132 of *Lecture Notes in Computer Science*, pages 132–146. Springer, 2011.

[20] E. Black and A. Hunter. An inquiry dialogue system. *Autonomous Agents and Multi-Agent Systems*, 19(2):173–209, 2009.

[21] G. Boella, D.M. Gabbay, A. Perotti, L. van der Torre, and S. Villata. Conditional labelling for abstract argumentation. In S. Modgil, N. Oren, and F. Toni, editors, *Theory and Applications of Formal Argumentation - First International Workshop, TAFA 2011*, volume 7132 of *Lecture Notes in Computer Science*, pages 232–248. Springer, 2011.

[22] A. Bondarenko, P.M. Dung, R.A. Kowalski, and F. Toni. An abstract, argumentation-theoretic approach to default reasoning. *Artificial Intelligenece*, 93, 1997.

[23] R.F. Brena, C.I. Chesñevar, and J.-L. Aguirre. Argumentation-supported information distribution in a multiagent system for knowledge management. In S. Parsons, N. Maudet, P. Moraitis, and I. Rahwan, editors, *Argumentation in Multi-Agent Systems, Second International Workshop, ArgMAS 2005*, volume 4049 of *Lecture Notes in Computer Science*, pages 279–296. Springer, 2005.

[24] M. Caminada. Semi-stable semantics. In P.E. Dunne and T.J.M. Bench-Capon, editors, *Computational Models of Argument: Proceedings of COMMA 2006*, volume 144 of *Frontiers in Artificial Intelligence and Applications*, pages 121–130. IOS Press, 2006.

[25] M. Caminada. An algorithm for computing semi-stable semantics. In K. Mellouli, editor, *Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 9th European Conference, ECSQARU 2007, Proceedings*, volume 4724 of *Lecture Notes in Computer Science*, pages 222–234. Springer, 2007.

[26] M. Caminada. Comparing two unique extension semantics for formal argumentation: ideal and eager. In M.M. Dastani and E. de Jong, editors, *19th Belgian-Dutch Conference on Artificial Intelligence, BNAIC 2007, Proceedings*, pages 81–87, 2007.

[27] M. Caminada. An algorithm for stage semantics. In P. Baroni, F. Cerutti, M. Giacomin, and G.R. Simari, editors, *Computational Models of Argument: Proceedings of COMMA 2010*, volume 216 of *Frontiers in Artificial Intelligence and Applications*, pages 147–158. IOS Press, 2010.

[28] C. Cayrol, S. Doutre, and J. Mengin. On decision problems related to the preferred semantics for argumentation frameworks. *Logic and Computation*, 13(3):377–403, 2003.

[29] C. Cayrol and M.-C. Lagasquie-Schiex. On the acceptability of arguments in bipolar argumentation frameworks. In L. Godo, editor, *Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 8th European Conference, ECSQARU*

2005, Proceedings, volume 3571 of *Lecture Notes in Computer Science*, pages 378–389. Springer, 2005.

[30] C.I. Chesñevar and G.R. Simari. A lattice-based approach to computing warranted beliefs in skeptical argumentation frameworks. In M.M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 280–285, 2007.

[31] S. Coste-Marquis, C. Devred, S. Konieczny, M.-C. Lagasquie-Schiex, and P. Marquis. On the merging of Dung's argumentation systems. *Artificial Intelligence*, 171(10-15):730–753, 2007.

[32] S. Coste-Marquis, C. Devred, and P. Marquis. Prudent semantics for argumentation frameworks. In *17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2005)*, pages 568–572. IEEE Computer Society, 2005.

[33] Y. Dimopoulos, V. Magirou, and C. Papadimitriou. On kernels, defaults and even graphs. *Annals of Mathematics and Artificial Intelligence*, 20:1–12, 1997.

[34] Y. Dimopoulos, B. Nebel, and F. Toni. Finding admissible and preferred arguments can be very hard. In A.G. Cohn, F. Giunchiglia, and B. Selman, editors, *KR 2000, Principles of Knowledge Representation and Reasoning Proceedings of the Seventh International Conference*, pages 53–61. Morgan Kaufmann, 2000.

[35] S. Doutre and J. Mengin. Preferred extensions of argumentation frameworks: Query answering and computation. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Automated Reasoning, First International Joint Conference, IJCAR 2001, Proceedings*, volume 2083 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2001.

[36] S. Doutre and J. Mengin. On sceptical versus credulous acceptance for abstract argument systems. In J.J. Alferes and J.A. Leite, editors, *Logics in Artificial Intelligence, 9th European Conference, JELIA 2004, Proceedings*, volume 3229 of *Lecture Notes in Computer Science*, pages 462–473. Springer, 2004.

[37] P.M. Dung. On the acceptability of arguments and its fundamental role in non monotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–357, 1995.

[38] P.M. Dung, R.A. Kowalski, and F. Toni. Dialectic proof procedures for assumption-based, admissible argumentation. *Artificial Intelligenece*, 170(2):114–159, 2006.

[39] P.M. Dung, P. Mancarella, and F. Toni. Computing ideal skeptical argumentation. *Artificial Intelligence*, 171(10-15):642–674, 2007.

[40] P.E. Dunne. Computational properties of argument systems satisfying graph-theoretic constraints. *Artificial Intelligence*, 171:701–729, 2007.

[41] P.E. Dunne. The computational complexity of ideal semantics. *Artificial Intelligence*, 173(18):1559 – 1591, 2009.

[42] P.E. Dunne. Tractability in value-based argumentation. In P. Baroni, F. Cerutti, M. Giacomin, and G.R. Simari, editors, *Computational Models of Argument: Proceedings of COMMA 2010*, volume 216 of *Frontiers in Artificial Intelligence and Applications*, pages 195–206. IOS Press, 2010.

[43] P.E. Dunne and T.J.M. Bench-Capon. Two party immediate response disputes: Properties and efficiency. *Artificial Intelligence*, 149(2):221–250, 2003.

[44] P.E. Dunne, A. Hunter, P. McBurney, S. Parsons, and M. Wooldridge. Weighted argument systems: Basic definitions, algorithms, and complexity results. *Artificial Intelligence*, 175(2):457–486, 2011.

[45] P.E. Dunne, P. Marquis, and M. Wooldridge. Argument aggregation: Basic axioms and complexity results. In B. Verheij, S. Szeider, and S. Woltran, editors, *Computational Models of Argument - Proceedings of COMMA 2012*, volume 245 of *Frontiers in Artificial Intelligence and Applications*, pages 129–140. IOS Press, 2012.

[46] P.E. Dunne, D.C. Martínez, A.J. García, and G.R. Simari. Computation with varied-strength attacks in abstract argumentation frameworks. In P. Baroni, F. Cerutti, M. Giacomin, and G.R. Simari, editors, *Computational Models of Argument: Proceedings of COMMA 2010*, volume 216 of *Frontiers in Artificial Intelligence and Applications*, pages 207–218. IOS Press, 2010.

[47] P.E. Dunne, S. Modgil, and T.J.M. Bench-Capon. Computation in extended argumentation frameworks. In H. Coelho, R. Studer, and M. Wooldridge, editors, *ECAI 2010 - 19th European Conference on Artificial Intelligence, Proceedings*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 119–124. IOS Press, 2010.

[48] W. Dvorák, P.E. Dunne, and S. Woltran. Parametric properties of ideal semantics. In T. Walsh, editor, *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 851–856. IJCAI/AAAI, 2011.

[49] W. Dvorák and S.A. Gaggl. Computational aspects of cf2 and stage2 argumentation semantics. In B. Verheij, S. Szeider, and S. Woltran, editors, *Computational Models of Argument - Proceedings of COMMA 2012*, volume 245 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2012.

[50] W. Dvorák and S.A. Gaggl. Incorporating stage semantics in the scc-recursive schema for argumentation semantics. In *In Proceedings of the 14th International Workshop on Non-Monotonic Reasoning (NMR 2012)*, 2012.

[51] W. Dvorák, M. Järvisalo, J. Peter Wallner, and S. Woltran. Complexity-sensitive decision procedures for abstract argumentation. In G. Brewka, T. Eiter, and S.A. McIlraith, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012*. AAAI Press, 2012.

[52] W. Dvořák, R. Pichler, and S. Woltran. Towards fixed-parameter tractable algorithms for abstract argumentation. *Artificial Intelligence*, 186:1–37, 2012.

[53] W. Dvořák, S.A. Gaggl, J. Wallner, and S. Woltran. Making use of advances in answer-set programming for abstract argumentation systems. Technical Report DBAI-TR-2011-70, Technische Universitt Wien, 2011.

[54] V. Efstathiou and A. Hunter. Focused search for arguments from propositional knowledge. In P. Besnard, S. Doutre, and A. Hunter, editors, *Computational Models of Argument: Proceedings of COMMA 2008*, volume 172 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2008.

[55] V. Efstathiou and A. Hunter. Algorithms for generating arguments and counterarguments in propositional logic. *International Journal of Approximate Reasoning*, 52(6):672–704, 2011.

[56] U. Egly, S.A. Gaggl, and S. Woltran. Answer-set programming encodings for argumentation frameworks. *Argument and Computation*, 1(2):147–177, 2010.

[57] D.M. Gabbay. Semantics for higher level attacks in extended argumentation frames part 1: Overview. *Studia Logica*, 93(2-3):357–381, 2009.

[58] D. Gaertner and F. Toni. Computing arguments and attacks in assumption-based argumentation. *IEEE Intelligent Systems*, 22(6):24–33, 2007.

[59] D. Gaertner and F. Toni. Hybrid argumentation and its properties. In P. Besnard, S. Doutre, and A. Hunter, editors, *Computational Models of Argument: Proceedings of COMMA 2008*, volume 172 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2008.

[60] A. Garcez, D.M. Gabbay, and L. Lamb. Value-based argumentation frameworks as neural-symbolic learning systems. *J. Log. Comput.*, 15(6):1041–1058, 2005.

[61] A.J. GARCIA and G.R. SIMARI. Defeasible logic programming: an argumentative approach. *Theory and Practice of Logic Programming*, 4:95–138, 0 2004.

[62] A. Hunter and M. Williams. Aggregating evidence about the positive and negative effects of treatments. *Artificial Intelligence in Medicine*, 56(3):173–190, 2012.

[63] H. Jakobovits and D. Vermeir. Dialectic semantics for argumentation frameworks. In *Proceedings of the International Conference on Artificial Intelligence and Law, ICAIL 1999*, pages 53–62, 1999.

[64] H. Jakobovits and D. Vermeir. Robust semantics for argumentation frameworks. *J. Log. Comput.*, 9(2):215–261, 1999.

[65] S. Kaci and L. van der Torre. Preference-based argumentation: Arguments supporting multiple values. *International Journal of Approximate Reasoning*, 48(3):730 – 751, 2008.

[66] A.C. Kakas and F. Toni. Computing argumentation in logic programming. *J. Log. Comput.*, 9(4):515–562, 1999.

[67] E.J. Kim and S. Ordyniak. Valued-based argumentation for tree-like value graphs. In B. Verheij, S. Szeider, and S. Woltran, editors, *Computational Models of Argument - Proceedings of COMMA 2012*, volume 245 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2012.

[68] E.J. Kim, S. Ordyniak, and S. Szeider. Algorithms and complexity results for persuasive argumentation. *Artificial Intelligence*, 175:1722–1736, 2011.

[69] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The dlv system for knowledge representation and reasoning. *ACM Trans. Computational Logic*, 7(3):499–562, 2006.

[70] H. Li, N. Oren, and T.J. Norman. Probabilistic argumentation frameworks. In S. Modgil, N. Oren, and F. Toni, editors, *Theory and Applications of Formal Argumentation - First International Workshop, TAFA 2011*, volume 7132 of *Lecture Notes in Computer Science*. Springer, 2011.

[71] B.S. Liao, L. Jin, and R.C. Koons. Dynamics of argumentation systems: A division-based method. *Artificial Intelligence*, 175(11):1790–1814, 2011.

[72] D.C. Martínez, A.J. García, and G.R. Simari. An abstract argumentation framework with varied-strength attacks. In G. Brewka and J. Lang, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference, KR 2008*, pages 135–144. AAAI Press, 2008.

[73] P. McBurney and S. Parsons. Dialogue games for agent argumentation. In I. Rahwan and G.R. Simari, editors, *Argumentation in Artificial Intelligence*, pages 261–280. Springer, 2009.

[74] C.C. McGeoch. Experimental algorithmics. *Communications of the ACM*, 50(11):27–31, 2007.

[75] S. Modgil. Labellings and games for extended argumentation frameworks. In C. Boutilier, editor, *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence*, 2009.

[76] S. Modgil. Reasoning about preferences in argumentation frameworks. *Artificial Intelligence*, 173:901–934, 2009.

[77] S. Modgil and T.J.M. Bench-Capon. Metalevel argumentation. *J. Log. Comput.*, 21(6):959–1003, 2011.

[78] S. Modgil and M. Caminada. Proof theories and algorithms for abstract argumentation frameworks. In I. Rahwan and G.R. Simari, editors, *Argumentation in Artificial Intelligence*, pages 105–129. Springer, 2009.

[79] M. Mozina, J. Zabkar, and I. Bratko. Argument based machine learning. *Artificial Intelligence*, 171:922–937, 2007.

[80] S.H. Nielsen and S. Parsons. Computing preferred extensions for argumentation systems with sets of attacking arguments. In P.E. Dunne and T.J.M. Bench-Capon, editors, *Computational Models of Argument: Proceedings of COMMA 2006*, volume 144 of *Frontiers in Artificial Intelligence and Applications*, pages 97–108. IOS Press, 2006.

[81] S.H. Nielsen and S. Parsons. A generalization of dung's abstract framework for argumentation: Arguing with sets of attacking arguments. In N. Maudet, S. Parsons, and I. Rahwan, editors, *Argumentation in Multi-Agent Systems, Third International Workshop, ArgMAS 2006*, volume 4766 of *Lecture Notes in Computer Science*. Springer, 2006.

[82] J.C. Nieves, U. Cortes, and M. Osorio. Preferred extensions as stable models. *Theory and Practice of Logic Programming*, 8(4):527–543, 2008.

[83] S. Nofal, P.E. Dunne, and K. Atkinson. On preferred extension enumeration in abstract argumentation. In B. Verheij, S. Szeider, and S. Woltran, editors, *Computational Models of Argument - Proceedings of COMMA 2012*, volume 245 of *Frontiers in Artificial Intelligence and Applications*, pages 205–216. IOS Press, 2012.

[84] S. Nofal, P.E. Dunne, and K. Atkinson. Towards average-case algorithms for abstract argumentation. In J. Filipe and A.L.N. Fred, editors, *ICAART (1)*, pages 225–230. SciTePress, 2012.

[85] S. Nofal, P.E. Dunne, and K. Atkinson. Towards experimental algorithms for abstract argumentation. In B. Verheij, S. Szeider, and S. Woltran, editors, *Computational Models of Argument - Proceedings of COMMA 2012*, volume 245 of *Frontiers in Artificial Intelligence and Applications*, pages 217–228. IOS Press, 2012.

[86] S. Nofal, P.E. Dunne, and K. Atkinson. Algorithms for acceptance in argument systems. In J. Filipe and A.L.N. Fred, editors, *ICAART 2013 - Proceedings of the 5th International Conference on Agents and Artificial Intelligence, Volume 1 - Artificial Intelligence*, pages 34–43. SciTePress, 2013.

[87] S. Ordyniak and S. Szeider. Augmenting tractable fragments of abstract argumentation. In T. Walsh, editor, *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 1033–1038. IJCAI/AAAI, 2011.

[88] H. Prakken. An abstract framework for argumentation with structured arguments. *Argument and Computation*, 1(2):93–124, 2010.

[89] H. Prakken. Formalising a legal opinion on a legislative proposal in the aspic$^+$ framework. In B. Schäfer, editor, *Legal Knowledge and Information Systems - JURIX 2012: The Twenty-Fifth Annual Conference*, volume 250 of *Frontiers in Artificial Intelligence and Applications*, pages 119–128. IOS Press, 2012.

[90] I. Rahwan and G.R. Simari. *Argumentation in Artificial Intelligence*. Springer, 2009.

[91] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1-2):81–132, 1980.

[92] N.D. Rotstein, S. Gottifredi, A.J. García, and G.R. Simari. A heuristics-based pruning technique for argumentation trees. In S. Benferhat and J. Grant, editors, *Scalable Uncertainty Management - 5th International Conference, SUM 2011*, volume 6929 of *Lecture Notes in Computer Science*. Springer, 2011.

[93] N.D. Rotstein, M.O. Moguillansky, A.J. García, and G.R. Simari. A dynamic argumentation framework. In P. Baroni, F. Cerutti, M. Giacomin, and G.R. Simari, editors, *Computational Models of Argument: Proceedings of COMMA 2010*, volume 216 of *Frontiers in Artificial Intelligence and Applications*, pages 427–438. IOS Press, 2010.

[94] N.D. Rotstein, M.O. Moguillansky, and G.R. Simari. Dialectical abstract argumentation: A characterization of the marking criterion. In C. Boutilier, editor, *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 898–903, 2009.

[95] P.M. Thang, P.M. Dung, and N.D. Hung. Towards a common framework for dialectical proof procedures in abstract argumentation. *J. Log. Comput.*, pages 1071–1109, 2009.

[96] B. Verheij. Two approaches to dialectical argumentation: admissible sets and argumentation stages. In *The Eighth Dutch Conference on AI*, pages 357–368, 1996.

[97] B. Verheij. A labeling approach to the computation of credulous acceptance in argumentation. In M.M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 623–628, 2007.

[98] S. Villata, G. Boella, and L. van der Torre. Attack semantics for abstract argumentation. In T. Walsh, editor, *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 406–413. IJCAI/AAAI, 2011.

[99] G. Vreeswijk. An algorithm to compute minimally grounded and admissible defence sets in argument systems. In P.E. Dunne and T.J.M. Bench-Capon, editors, *Computational Models of Argument: Proceedings of COMMA 2006*, volume 144 of *Frontiers in Artificial Intelligence and Applications*, pages 109–120. IOS Press, 2006.

[100] G. Vreeswijk and H. Prakken. Credulous and sceptical argument games for preferred semantics. In M. Ojeda-Aciego, I.P. de Guzmán, G. Brewka, and L.M. Pereira, editors, *Logics in Artificial Intelligence, European Workshop, JELIA 2000, Proceedings*, volume 1919 of *Lecture Notes in Computer Science*, pages 239–253. Springer, 2000.